Finding near-duplicates

Mining Massive Datasets

Carlos Castillo

Topic 04

Universitat Pompeu Fabra Barcelona

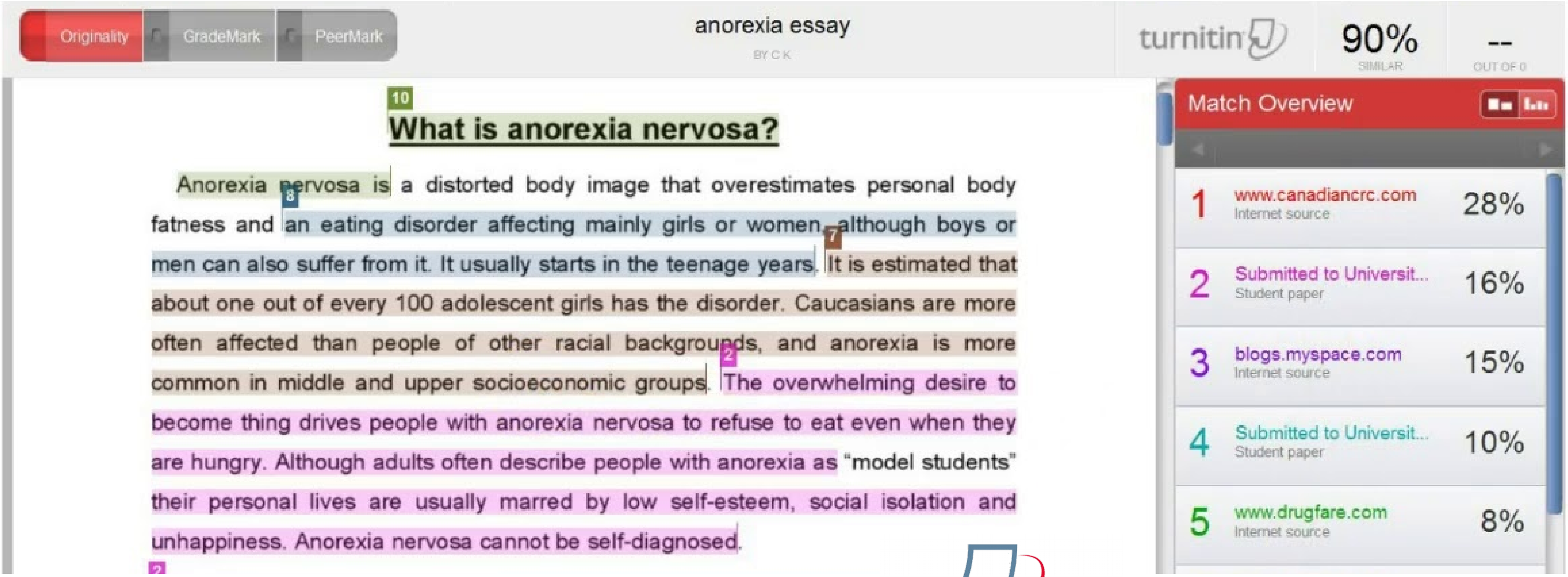
Source for this deck

- Mining of Massive Datasets 2nd edition (2014) by Leskovec et al. (Chapter 3) [slides ch3]

Fast near-neighbor applications

- For documents
  - Find “legitimate” duplicates
    - Copies of the same press release or cable
    - Mirrors of the same documents, for efficiency
  - Find “illegitimate” duplicates
    - Plagiarism
- For baskets
  - Find customers who purchase similar items

Example: plagiarism detection



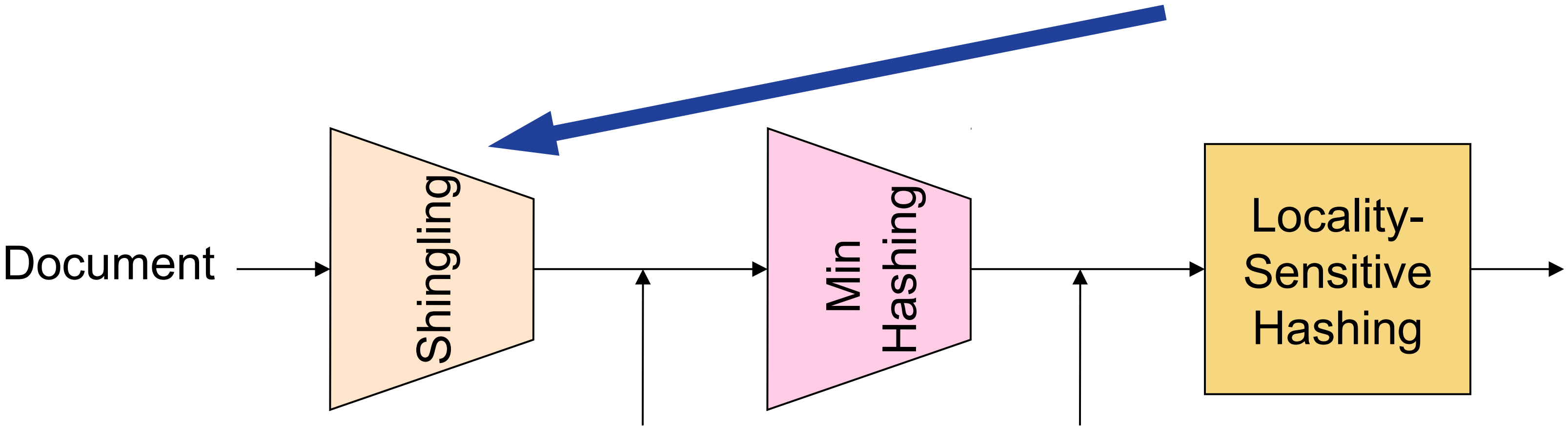
turnitin

Fast near-neighbor challenges

- Too many documents to compare all pairs
  - OK to pay linear or log cost, but not quadratic
- Documents cannot fit in main memory
  - They are too large or too many
- Many small pieces of one document can appear out of order in another

Shingling
(ngrams)

First step: shingling



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

The set of strings of length ***k*** that appear in the document

Signatures: short integer vectors that represent the sets, and reflect their similarity

Naïve solution:
feature selection over bag of words

- Document = set of terms

  → Document = set of important terms

- Now, compute all pairs similarity
- Doesn't work for at least two reasons, why?

Naïve solution: feature selection over bag of words

- Document = set of terms

  → Document = set of important terms

- Now, compute all pairs similarity
- Doesn't work for at least two reasons, why?
  - Doesn't preserve the ordering
  - Unimportant terms are also relevant (stylistic)

Shingles

- An ***ngram*** in a document is a sequence of n tokens that appears in the doc
- **Shingles** are either ngrams (word-level) or sequences of characters, depending on the application
- **Character-level example:** **k=2**; document $\mathbf{D_1}$ = abcab
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}
  - **Option:** Shingles as a bag (multiset), count ab twice: $\mathbf{S'(D_1)}$ = {ab, bc, ca, ab}

Example: 4-grams (shingle = 4 consecutive words)

E.g., 4-shingles of
"`My name is Inigo Montoya. You killed my father. Prepare to die`":

{

- my name is inigo
- name is inigo montoya
- is inigo montoya you
- inigo montoya you killed
- montoya you killed my
- you killed my father
- killed my father prepare
- my father prepare to
- father prepare to die

}

Compressed representation of shingles

- To **compress long shingles**, we can **hash** them to (say) 4 bytes
- **Represent a document by the set of hash values of its *k*-shingles**
- **Idea:** Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared

- **Example:** **k=2**; document $\mathbf{D_1}$= abcab
  Set of 2-shingles: $\mathbf{S(D_1)}$ = {ab, bc, ca}
  Hash the singles: $\mathbf{h(D_1)}$ = {1, 5, 7}

Documents as sets of shingles

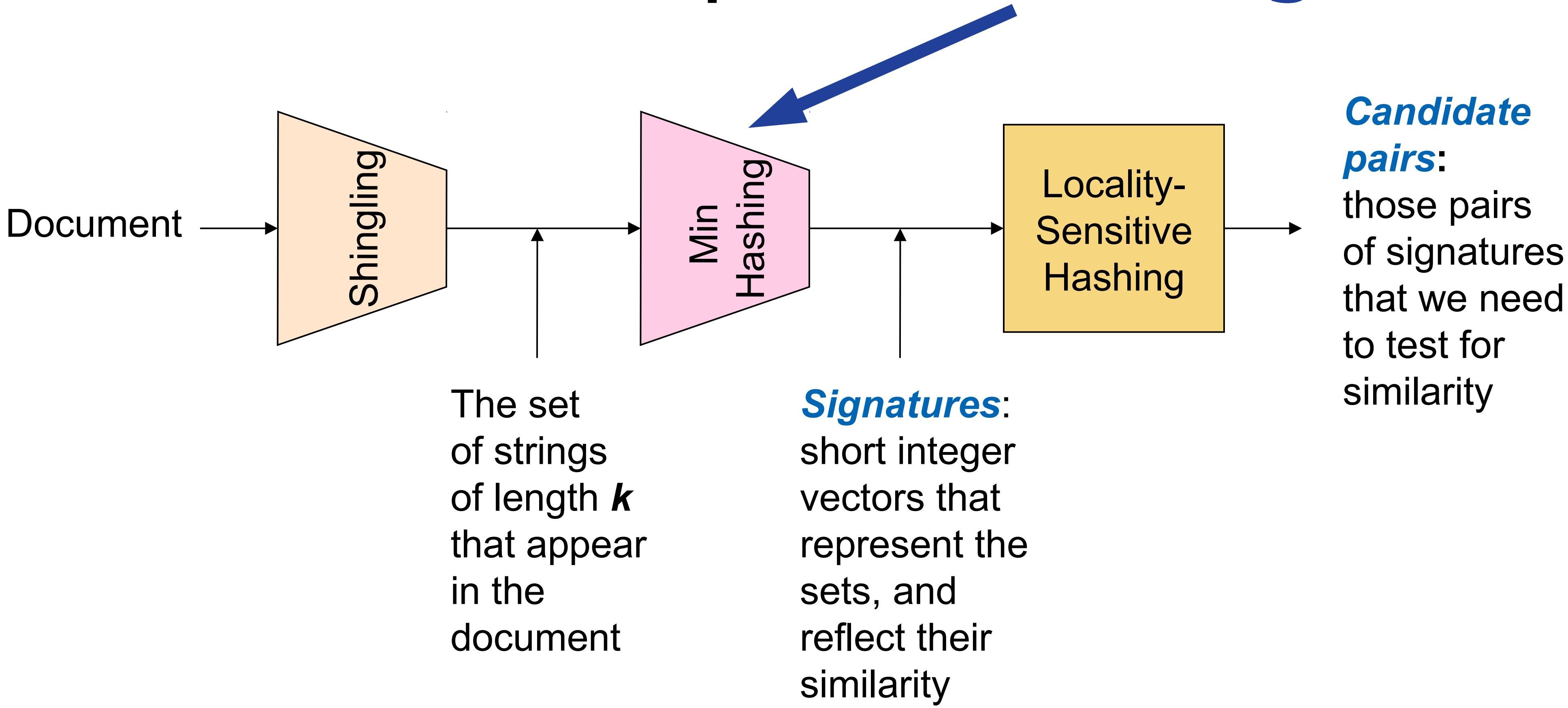
- A document is now a set of shingles
  - Dimensionality reduced from “words in a dictionary” to “number of distinct shingles”
  - Higher dimensionality but more sparse
- Working assumption
  - Documents that have lots of shingles in common have similar text, even if the text appears in different order
- Caveat: You must pick k large enough, or most documents will have most shingles
  - k = 5 is OK for short documents
  - k = 10 is better for long documents

Using shingles directly

- **Suppose we need to find near-duplicate documents among million documents**
- Naïvely, we would have to compute **all pairwise Jaccard similarities** ≈ **$5*10^{11}$** comparisons
- At 10^5 secs/day and 10^6 comparisons/sec, it would take **5 days**
- For 10 million, it takes more than a year…

Min hashing

Next step: min hashing



Sets can be bit vectors

- Many similarity problems involve **finding subsets with substantial intersection**
- **Remember we can encode sets using bit vectors**
  - set intersection = bitwise **AND**
  - set union = bitwise **OR**

$$J(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

- **Example:** $\mathbf{C_1}$ = 10111; $\mathbf{C_2}$ = 10011
  - Size of intersection **= 3**; size of union **= 4,**
  - **Jaccard similarity** (not distance) **= 3/4**
  - **Distance: d(C_1,C_2) = 1 – (Jaccard similarity) = 1/4**

From sets to boolean matrices

- **Rows = items** (shingles)
- **Columns = sets** (documents)
  - 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 corresponding sets (rows with value 1)
- Typical matrix is very sparse!

Documents

Shingles				
	1	1	1	0
	1	1	0	1
	0	1	0	1
	0	0	0	1
	1	0	0	1
	1	1	1	0
	1	0	1	0

Hashing set representations

- We don't want to compare c_1, c_2, they might be too large, slowing down the computation
- Instead, we compute signatures $h(c_1)$, $h(c_2)$ that are smaller in size than c_1 and c_2
- **Desired properties:**
  $c_1 = c_2 \Rightarrow \text{Prob.}(h(c_1) = h(c_2))$ is large
  $c_1 \neq c_2 \Rightarrow \text{Prob.}(h(c_1) \neq h(c_2))$ is large

Hashing set representations (cont.)

- Naïve approach (non-LSH-based):
  - 1) Compute signatures of columns: small summaries of columns
  - 2) Examine all pairs of signatures to find similar columns
    - Essential: Similarities of signatures and columns are related
  - 3) Optional: verify that columns with similar signatures are really similar
- Warnings:
  - Comparing all pairs may take too much time: Job for LSH
  - These methods can produce false negatives, and even false positives (if the optional check is not made)

Hash function for Jaccard metric: min hashing

- Imagine the rows of the boolean matrix permuted under **random but fixed permutation π**
- Define a **"hash" function $h_\pi(C)$** = the index of the **first** (in the permuted order π) row in which column ***C*** has value **1**:
  - $h_\pi(C) = min_\pi\ \pi(C)$
- Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column

Minhash example

Permutations π

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

Rows=Shingles, Columns=Documents

D1	D2	D3	D4
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*

D1	D2	D3	D4
2	1	2	1
2	1	4	1
1	2	1	2

4th element of the permutation is the first to map to a 1

Try it! Minhash

Permutation π | **Rows=Shingles, Columns=Documents**

π	D1	D2	D3	D4
3	1	0	1	0
2	1	0	0	1
1	0	1	0	1
4	0	1	0	1
7	0	1	0	1
5	1	0	1	0
6	1	0	1	0

Signature matrix *M*

D1	D2	D3	D4

Index of the bit vector position where the first 1 occurs according to the ordering of the permutation

Minhash approximates Jaccard

- **Choose a random permutation π**
- **<u>Claim:</u> $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$**
- **Why?**
  - Let **X** be a doc (set of shingles), ***y*∈*X*** is a shingle
  - **Then: $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$**
  - It is equally likely that any ***y*∈*X*** is mapped to the ***min*** element
  - Let ***y*** be s.t. $\pi(y) = \min(\pi(C_1 \cup C_2))$
  - **Then either:**

    $\pi(y) = \min(\pi(C_1))$ if $y \in C_1$ **or** $\pi(y) = \min(\pi(C_2))$ if $y \in C_2$
  - So the prob. that **both** are true is the prob. ***y*** $\in C_1 \cap C_2$
  - **$\Pr[\min(\pi(C_1))=\min(\pi(C_2))]=|C_1 \cap C_2|/|C_1 \cup C_2|= sim(C_1, C_2)$**

A single hash function is too coarse for our purposes

- We will use many permutations (say, 100)
- **A signature is a collection of minhashes:** one for each permutation
- Jaccard(c_1, c_2) = E[minhashsim(c_1, c_2)]
  - minhashsim(c1,c2) = J(minhash(C1), minhash(C2))

Example: three permutations

Permutation π

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

Rows=Shingles, Columns=Documents

D1	D2	D3	D4
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*

D1	D2	D3	D4
2	1	2	1
2	1	4	1
1	2	1	2

Similarities:

	1-3	2-4	1-2	3-4
Complete	0.75	0.75	0	0
Signatures	0.67	1.00	0	0

Minhash signatures

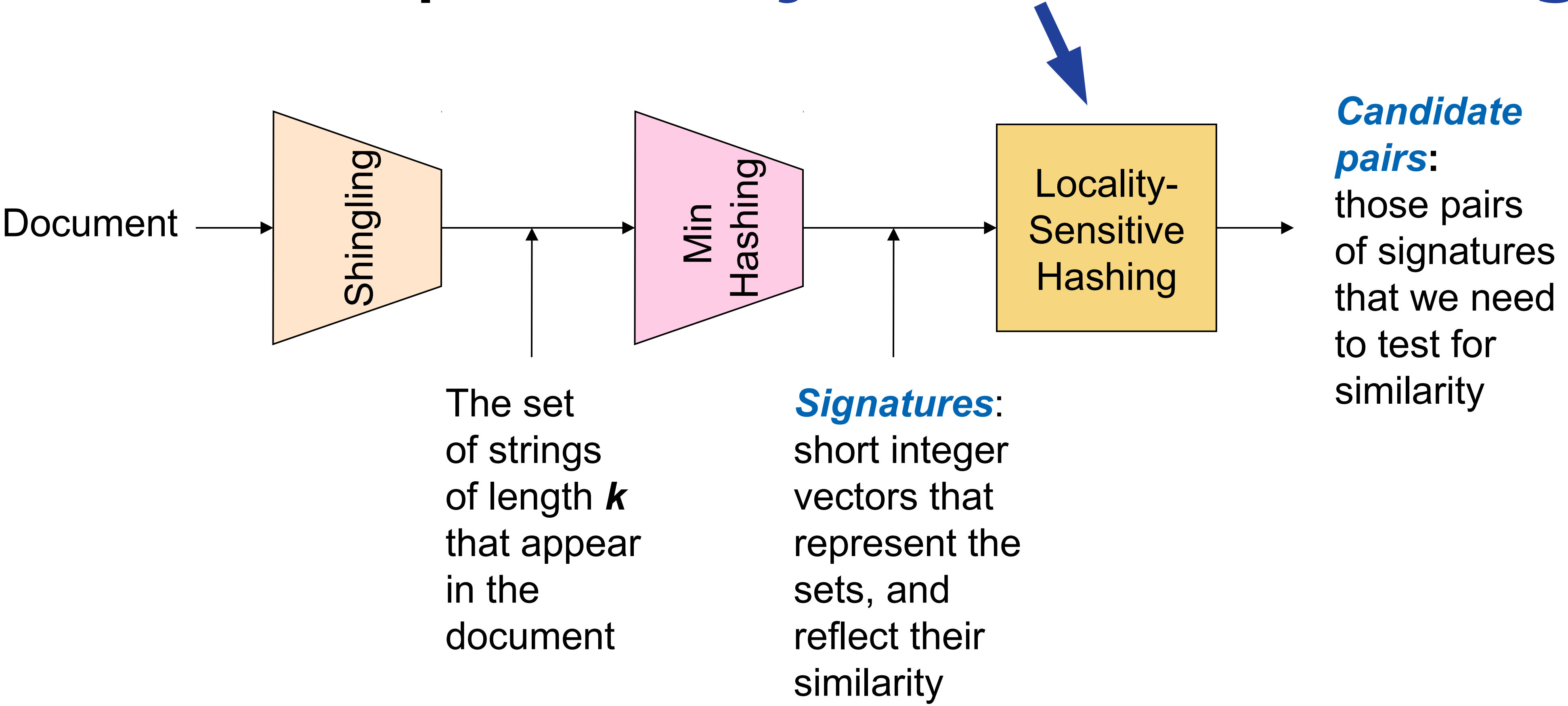
- **Pick $\pi_1 \ldots \pi_{100}$ random permutations of the rows (K=100)**
- Think of ***sig*(C)** as a column vector
  - ***sig*(C)[i] =** according to the *i*-th permutation, the index of the first row that has a 1 in column *C*
  - $\boldsymbol{sig(C)[i] = \min(\pi_i(C))}$
- The signature or "sketch" of document *C* has fixed size!
  - We achieved our goal: we "compressed" long bit vectors into short signatures

Implementation

- **Permuting rows even once is prohibitive**
- Create $\pi_1 \ldots \pi_{100}$ by using **K = 100** hash functions k_i
  - Ordering of {1,2,…,n} under k_i (computing h(1), h(2), …, h(n) and sorting in increasing order) gives a random permutation!
- **One-pass implementation**
  - For each column C and hash function k_i keep a variable for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - **Keep the min hash value in a row containing a 1:**
    - Suppose row j has 1 in column C
      - Then for each k_i If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

Locality-sensitive hashing

Final step: locality-sensitive hashing



LSH: first idea

- **Goal:** Find documents with Jaccard similarity at least ***s*** (for some similarity threshold, e.g., ***s***=0.8)
- **LSH – General idea:** Use a function ***f(x,y)*** that tells whether ***x*** and ***y*** is a ***candidate pair***: a pair of elements whose similarity must be evaluated
- **For Min-Hash matrices:**
  1) Hash columns of signature matrix ***M*** to many buckets
  2) Each pair of documents that hashes into the same bucket is a **candidate pair**

Signature matrix ***M***

d1	d2	d3	d4
2	1	4	1
1	2	1	2
2	1	2	1

Selecting candidates

- **Pick a similarity threshold *s* (0 < s < 1)**
- Columns ***x*** and ***y*** of ***M*** are a **candidate pair** if their signatures agree (***M* (*i*, *x*) = *M* (*i*, *y*)**) on at least fraction ***s*** of their rows
- We expect documents ***x*** and ***y*** to have the same (Jaccard) similarity as their signatures

Signature matrix *M*

d1	d2	d3	d4
2	1	4	1
1	2	1	2
2	1	2	1

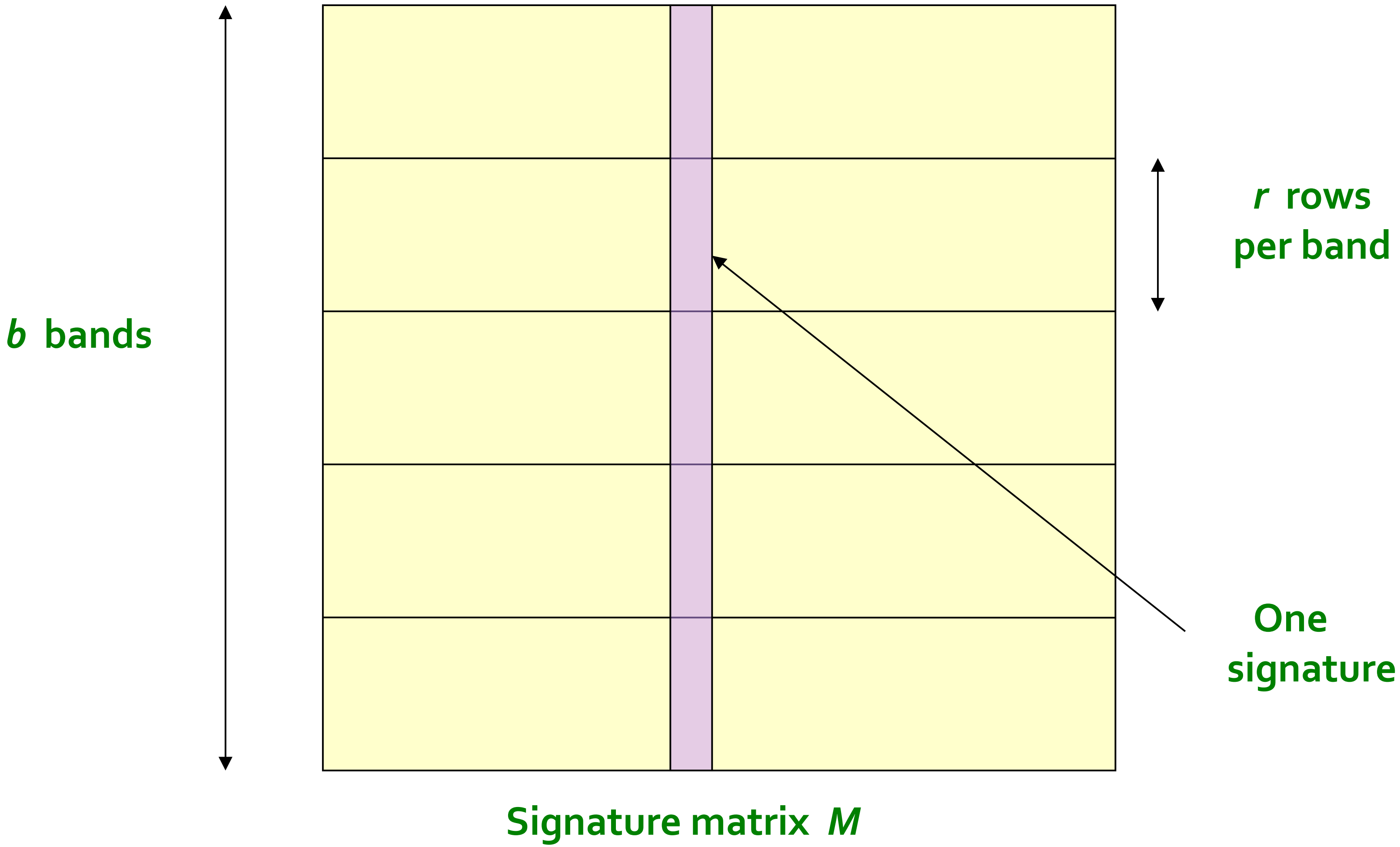
Creating buckets of similar documents

- **Big idea: hash columns of signature matrix *M***
- Arrange that (only) **similar columns** are likely to **hash to the same bucket**, with high probability
- **Candidate pairs are those that hash to the same bucket**

Signature matrix *M*

d1	d2	d3	d4
2	1	4	1
1	2	1	2
2	1	2	1

Partition M into b bands of size r



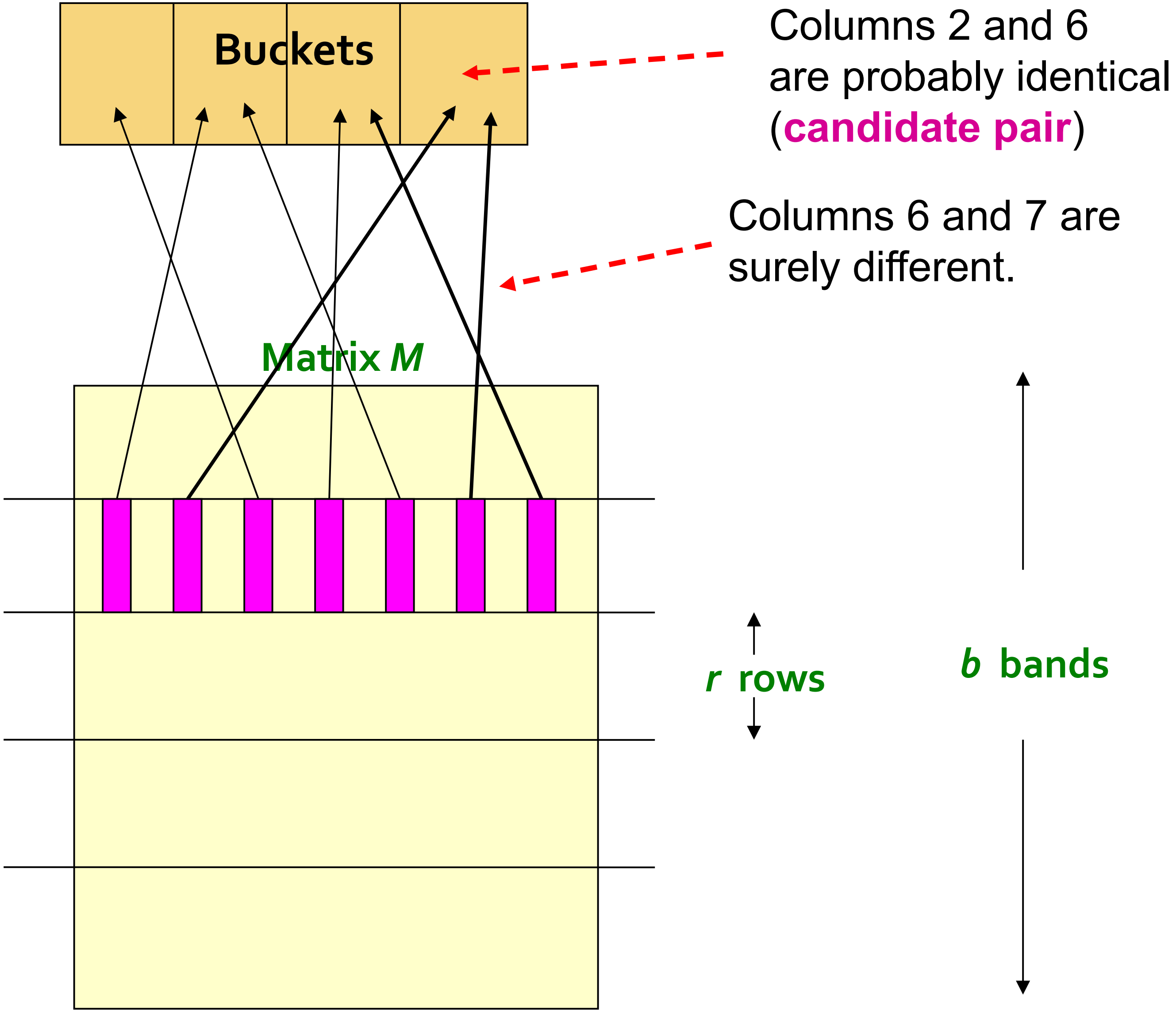
Partition M into b bands of size r (cont.)

- Partition 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 non-similar pairs

Signature matrix *M*

d1	d2	d3	d4
2	1	4	1
1	2	1	2
2	1	2	1

Hashing bands



Simplifying assumption: no collisions (no false positives)

- We will assume 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**"
- Assumption needed only to simplify analysis, not for correctness of algorithm

Example of bands

Assume the following case:

- Suppose 100,000 columns of ***M*** (100k docs)
- Signatures of 100 integers (rows)
  - Therefore, signatures take 40Mb
- Choose ***b*** = 20 bands of ***r*** = 5 integers/band
- **Goal:** Find pairs of documents that are at least ***s*** = *0.8* similar

Suppose sim(C_1, C_2) = 0.8

- **Find pairs of ≥ *s***=0.8 similarity, set **b**=20, **r**=5
- Since sim(C_1, C_2) $\geq$ **s**, we want C_1, C_2 to be a **candidate pair**
  - We want them to hash to **at least 1 common bucket** (at least one band is identical)
- **Probability C_1, C_2 identical in one particular band:** $(0.8)^5 = 0.328$
- Probability C_1, C_2 are ***not*** similar in all of the 20 bands:

  $(1\text{-}0.328)^{20} = 0.00035$
  - i.e., about 1/3000th of the 80%-similar column pairs are **false negatives** (we will miss them)
- **We would find 99.965% pairs of truly similar documents**

Suppose sim(C_1, C_2) = 0.3

- **Find pairs of ≥ *s*=0.8 similarity, set b=20, r=5**
- Since sim(C_1, C_2) < **s**, we **do not** want C_1, C_2 to be a **candidate pair**
- **Probability C_1, C_2 identical in one particular band:**

  $(0.3)^5 = 0.00243$
- Probability C_1, C_2 identical in at least 1 of 20 bands:

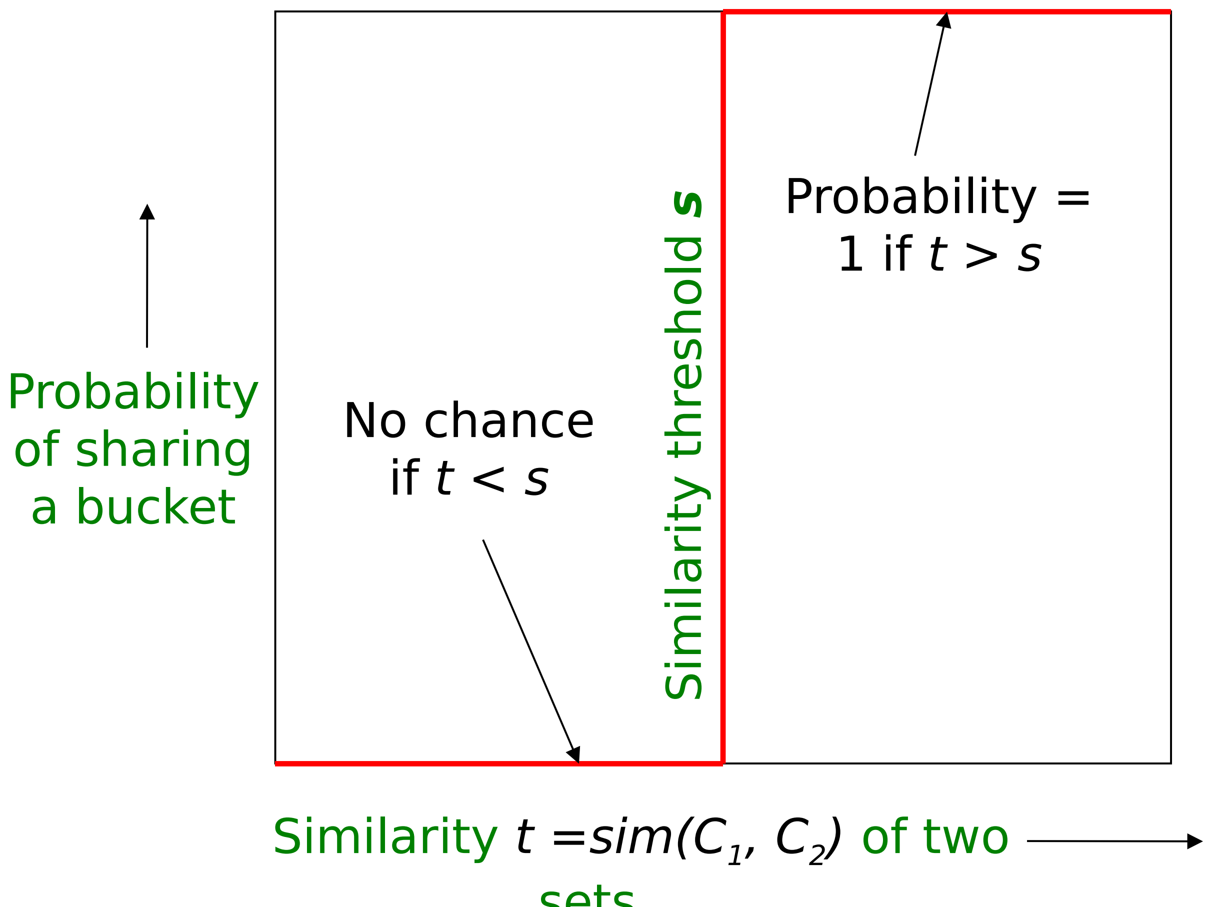
  $1 - (1 - 0.00243)^{20} = 0.0474$
- In other words, approximately 4.74% pairs of docs with similarity 0.3% end up becoming **candidate pairs**
  - They are **false positives** since we will have to examine them (they are candidate pairs) but then it will turn out their similarity is below threshold **s**

Advanced materials
(not for the exams) →

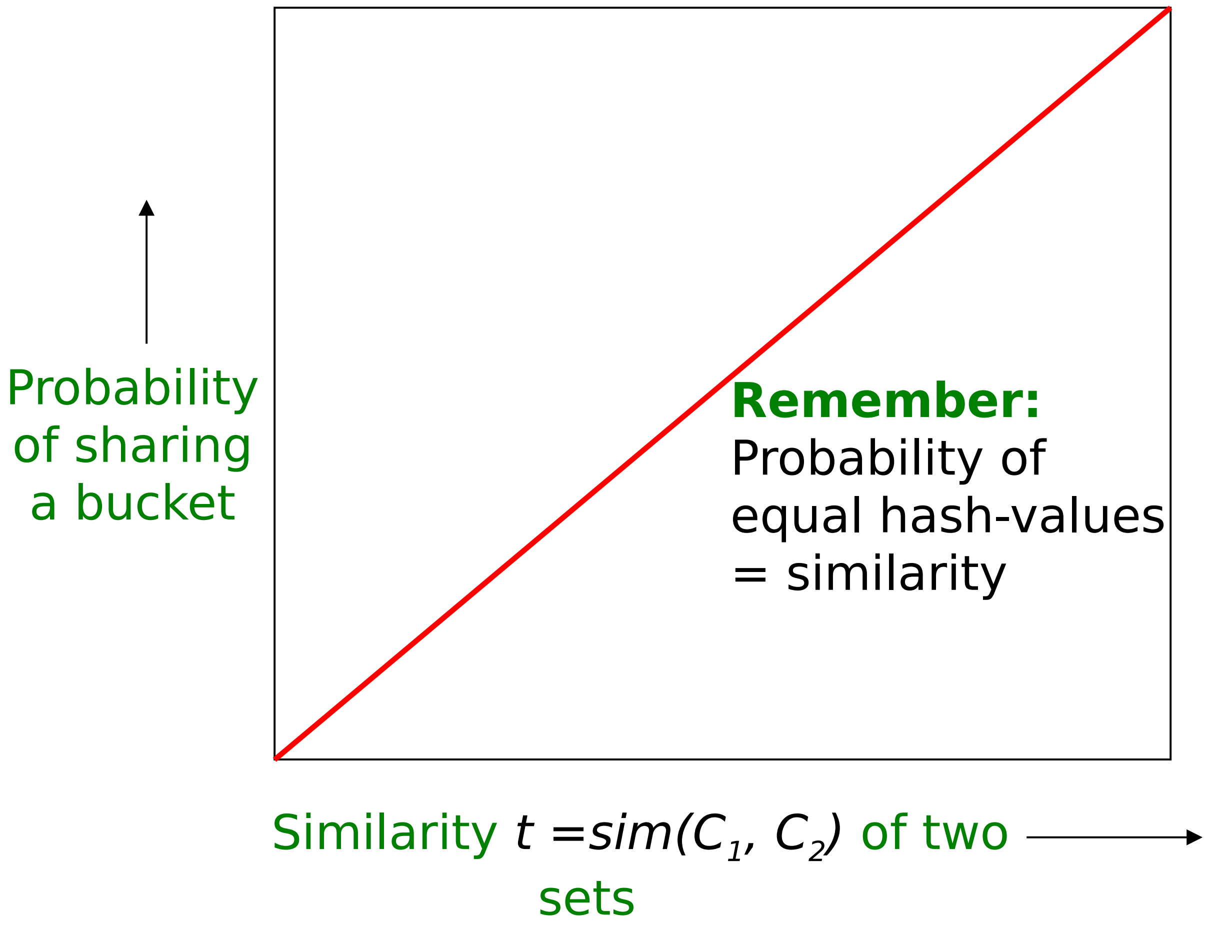
LSH involves a trade-off

- Pick:
  - The number of Min-Hashes (rows of M = K)
  - The number of bands b, and
  - The number of rows r per band to balance false positives/negatives
- Example: If we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up

LSH: what we want



What 1 band of 1 row gives you

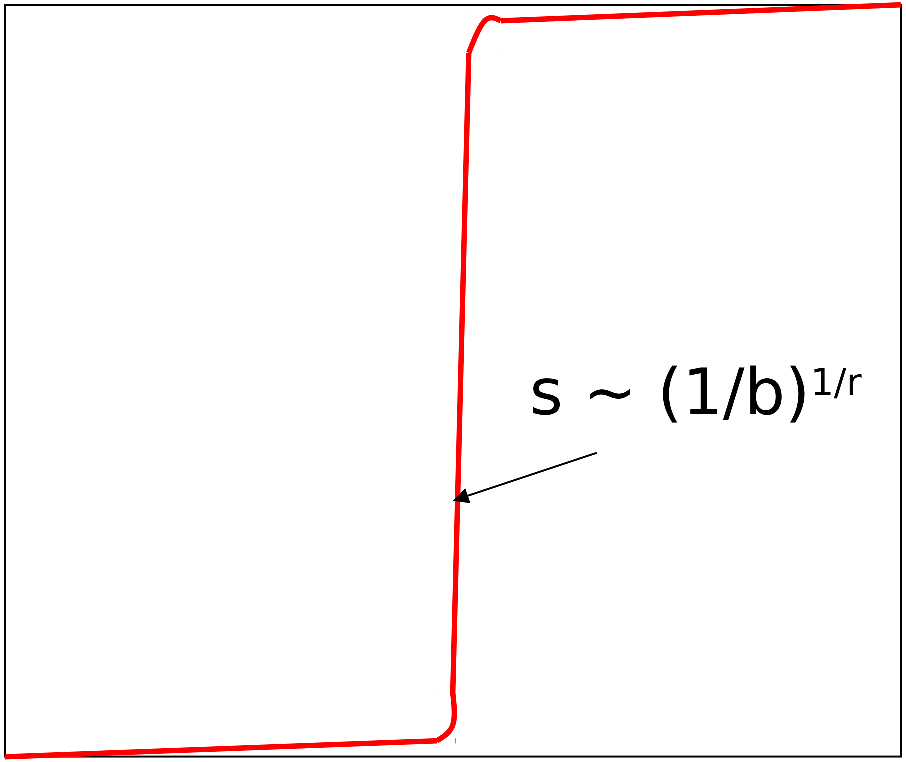


b bands, r rows/band

- Columns C_1 and C_2 have similarity $\boldsymbol{t}$
- Pick any band ($\boldsymbol{r}$ rows)
  - Prob. that all rows in band equal = $\boldsymbol{t^r}$
  - Prob. that some row in band unequal = $\mathbf{1 - t^r}$
- Prob. that no band identical = $\mathbf{(1 - t^r)^b}$
- Prob. that at least 1 band identical = $\mathbf{1 - (1 - t^r)^b}$

What b bands of r rows give you

Probability of sharing a bucket



Similarity $t=sim(C_1, C_2)$ of two sets

At least one band identical

No bands identical

$$\mathbf{1 - (1 - \mathit{t}^{\,r})^{b}}$$

Some row of a band unequal

All rows of a band are equal

Example: b=20, r=5

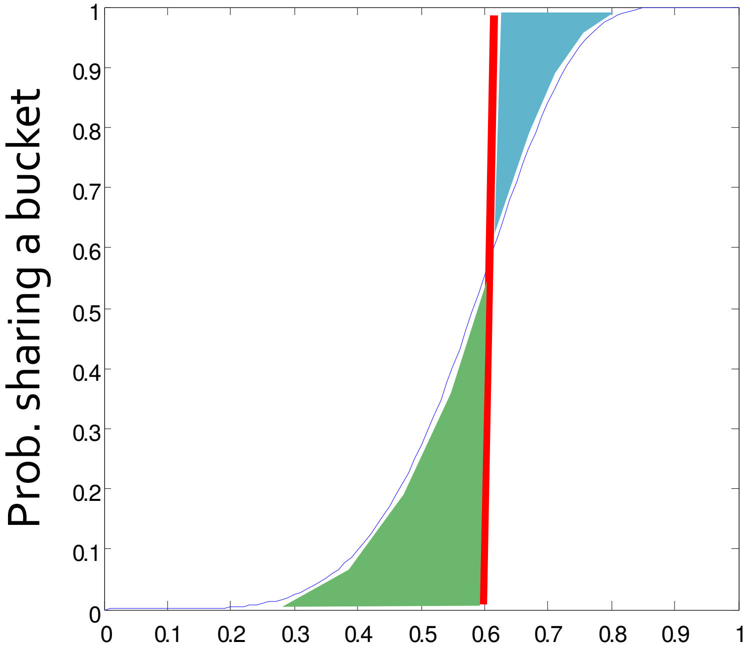
- **Similarity threshold s**
- **Prob. that at least 1 band is identical:**

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

Picking r and b: the S curve

Picking *r* and *b* to get the best S-curve

50 hash-functions (r=5, b=10)



Blue area: False Negative rate
Green area: False Positive rate

← End of advanced materials

LSH summary

- Tune ***K* (permutations), *b* (bands), *r* (permutations/ band)** to get almost all pairs with similar signatures, but eliminate most pairs that do not have similar signatures
- After finding candidates, check in main memory that **candidate pairs** really do have **similar signatures**

Summary

Things to remember

- **Shingling**: Convert documents to sets
  - We used hashing to assign each shingle an ID
- **Min-Hashing**: Convert large sets to short signatures, while preserving similarity
  - We used **similarity preserving hashing** to generate signatures with property $\mathbf{Pr}[h_\pi(C_1) = h_\pi(C_2)] = \mathbf{\mathit{sim}(C_1, C_2)}$
  - We used hashing to get around generating random permutations
- **Locality-Sensitive Hashing**: Focus on pairs of signatures likely to be from similar documents
  - We used hashing to find **candidate pairs** of similarity $\geq$ **s**

Exercises for this topic

- Mining of Massive Datasets 2nd edition (2014) by Leskovec et al.
  - Exercises 3.1.4 (Jaccard similarity)
  - Exercises 3.2.5 (Shingling)
  - Exercises 3.3.6 (Min hashing)
  - Exercises 3.4.4 (Locality-sensitive hashing)