Locality Sensitive Hashing

Vinay Setty
(vinay.j.setty@uis.no)

Slides credit: http://mmds.org

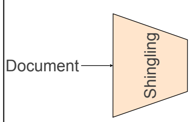
Finding Similar Items Problem

- Similar Items
- Finding similar web pages and news articles
- Finding near duplicate images
- Plagiarism detection
- Duplications in Web crawls
- Find nearest-neighbors in high-dimensional space
  - Nearest neighbors are points that are a small distance apart

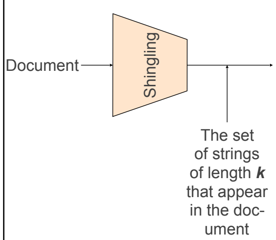
Very similar news articles

Near duplicate images

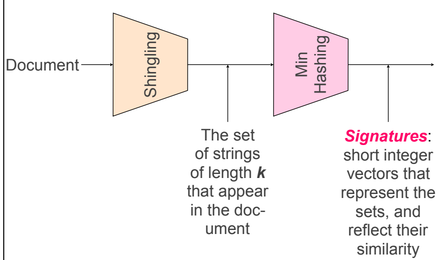
The Big Picture



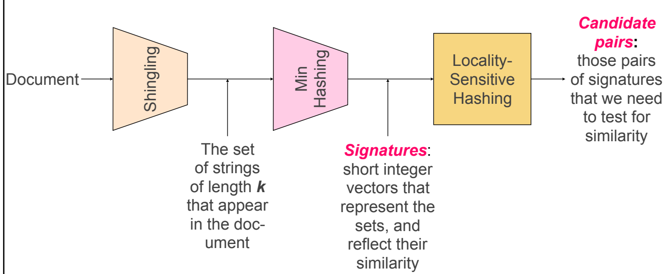
The Big Picture



The Big Picture



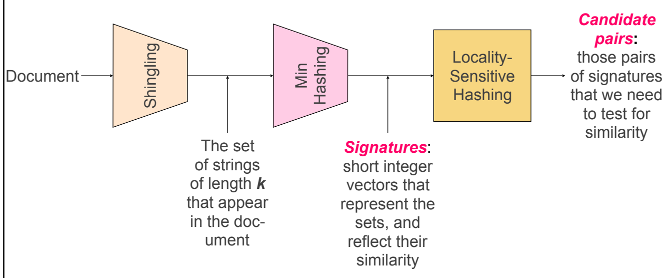
The Big Picture



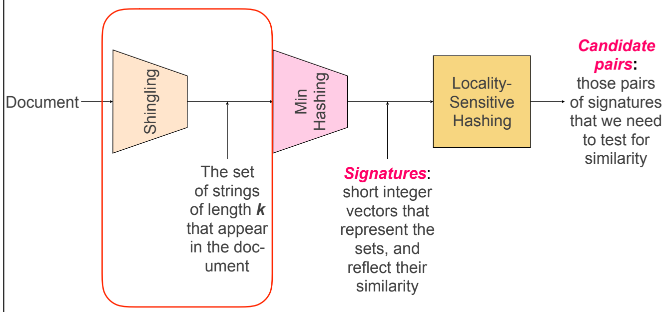
Three Essential Steps for Similar Docs

1. *Shingling:* Convert documents to sets
2. *Min-Hashing:* Convert large sets to short signatures, while preserving similarity
3. *Locality-Sensitive Hashing:* Focus on pairs of signatures likely to be from similar documents
   - **Candidate pairs!**

The Big Picture



The Big Picture



Documents as High-Dim. Data

Documents as High-Dim. Data

- **Step 1: *Shingling:* Convert documents to sets**

Documents as High-Dim. Data

- **Step 1: *Shingling:* Convert documents to sets**
- **Simple approaches:**
  - Document = set of words appearing in document
  - Document = set of "important" words
  - Don't work well for this application. Why?

Documents as High-Dim. Data

- **Step 1: *Shingling:* Convert documents to sets**
- **Simple approaches:**
  - Document = set of words appearing in document
  - Document = set of "important" words
  - Don't work well for this application. Why?
- **Need to account for ordering of words!**

Documents as High-Dim. Data

- **Step 1: *Shingling:* Convert documents to sets**
- **Simple approaches:**
  - Document = set of words appearing in document
  - Document = set of "important" words
  - Don't work well for this application. Why?
- **Need to account for ordering of words!**
- A different way: **Shingles!**

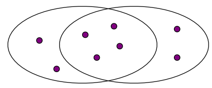
Define: Shingles

- A *k-shingle* (or *k-gram*) for a document is a sequence of *k* tokens that appears in the doc
  - Tokens can be characters, words or something else, depending on the application
  - Assume tokens = characters for examples
- **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}

Similarity Metric for Shingles

- **Document D_1 is a set of its k-shingles C_1=S(D_1)**
- Equivalently, each document is a 0/1 vector in the space of *k*-shingles
  - Each unique shingle is a dimension
  - Vectors are very sparse
- **A natural similarity measure is the Jaccard similarity:**

  $sim(D_1, D_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$



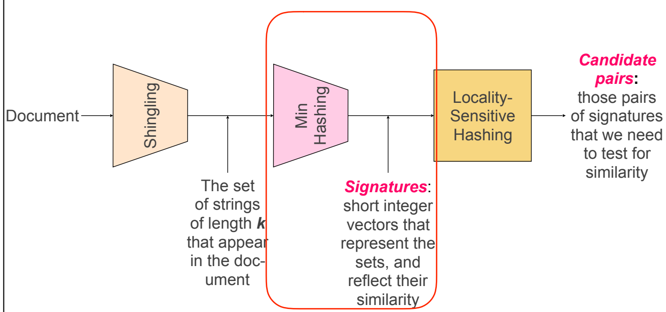
Working Assumption

- **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

Motivation for Minhash/LSH

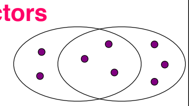
- **Suppose we need to find near-duplicate documents among $N = 1$ million documents**
- Naïvely, we would have to compute pairwise Jaccard similarities for **every pair of docs**
  - $N(N-1)/2 \approx 5*10^{11}$ comparisons
  - At 10^5 secs/day and 10^6 comparisons/sec, it would take **5 days**
- For $N = 10$ million, it takes more than a year…

The Big Picture



Encoding Sets as Bit Vectors

- Many similarity problems can be formalized as **finding subsets that have significant intersection**
- **Encode sets using 0/1 (bit, boolean) vectors**
  - One dimension per element in the universal set
- Interpret set intersection as bitwise **AND**, and set union as bitwise **OR**



- **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** = elements (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 sparse!**

From Sets to Boolean Matrices

- **Rows** = elements (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 sparse!**
- **Each document is a column:**
  - **Example: sim(C_1 ,C_2) = ?**
    - Size of intersection = 3; size of union = 6, Jaccard similarity (not distance) = 3/6
    - **d(C_1,C_2) = 1 – (Jaccard similarity) = 3/6**

Documents **(N)**

Shingles **(D)**

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 Columns (Signatures)

- **Key idea:** "hash" each column ***C*** to a small ***signature h(C)***, such that:
  - **(1)** ***h(C)*** is small enough that the signature fits in RAM
  - **(2)** $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$

Hashing Columns (Signatures)

- **Key idea:** "hash" each column ***C*** to a small ***signature h(C)***, such that:
  - **(1)** ***h(C)*** is small enough that the signature fits in RAM
  - **(2)** $sim(C_1, C_2)$ is the same as the "similarity" of signatures $h(C_1)$ and $h(C_2)$

- **Goal: Find a hash function $h(\cdot)$ such that:**
  - If $sim(C_1,C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
  - If $sim(C_1,C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$

- **Hash docs into buckets. Expect that "most" pairs of near duplicate docs hash into the same bucket!**

Min-Hashing

- **Goal:** Find a hash function $h(\cdot)$ such that:
  - if $sim(C_1,C_2)$ is high, then with high prob. $h(C_1) = h(C_2)$
  - if $sim(C_1,C_2)$ is low, then with high prob. $h(C_1) \neq h(C_2)$
- **Clearly, the hash function depends on the similarity metric:**
  - Not all similarity metrics have a suitable hash function
- **There is a suitable hash function for the Jaccard similarity:** It is called **Min-Hashing**

Min-Hashing

- Imagine the rows of the boolean matrix permuted under **random 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

Example

nput matrix (Shingles x Documents)
Permutation π

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

Example

nput matrix (Shingles x Documents)
Permutation π

π				
2	1	0	1	0
3	1	0	0	1
7	0	1	0	1
6	0	1	0	1
1	0	1	0	1
5	1	0	1	0
4	1	0	1	0

Example

nput matrix (Shingles x Documents)
Permutation π

π				
2	1	0	1	0
3	1	0	0	1
7	0	1	0	1
6	0	1	0	1
1	0	1	0	1
5	1	0	1	0
4	1	0	1	0

Signature matrix *M*

2	1	2	1

Example

2nd element of the permutation is the first to map to a 1

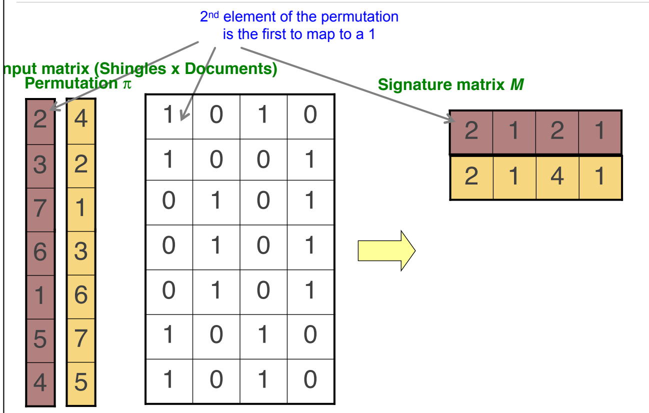
nput matrix (Shingles x Documents)
Permutation π

π				
2	1	0	1	0
3	1	0	0	1
7	0	1	0	1
6	0	1	0	1
1	0	1	0	1
5	1	0	1	0
4	1	0	1	0

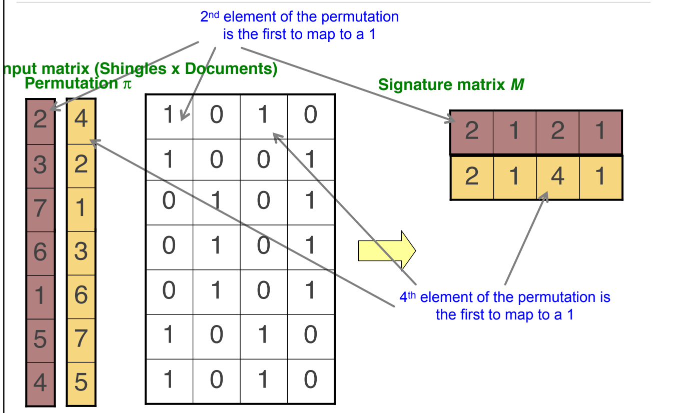
Signature matrix *M*

2	1	2	1

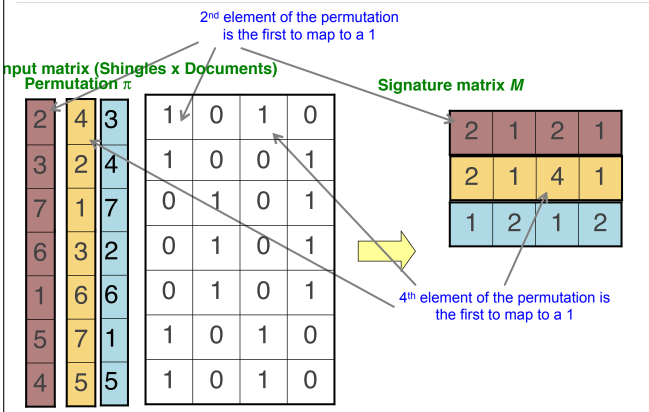
Example



Example



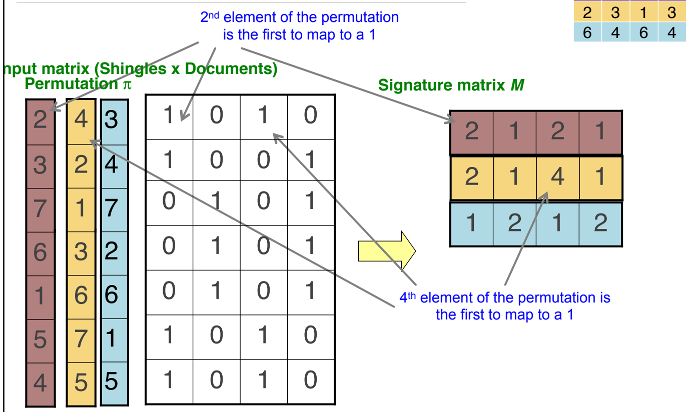
Example



Example

Note: Another (equivalent) way is to store row indexes:

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



Four Types of Rows

- **Given cols 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

  - **a** = # rows of type A, etc.
- **Note: sim(C_1, C_2) = a/(a +b +c)**
- **Then: Pr[$h(C_1) = h(C_2)$] = $Sim(C_1, C_2)$**
  - Look down the cols 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

Similarity for Signatures

- We know: **Pr[$h_\pi(C_1) = h_\pi(C_2)$] = *sim*(C_1, C_2)**

Similarity for Signatures

- We know: **Pr[$h_\pi(C_1) = h_\pi(C_2)$] = *sim*(C_1, C_2)**
- Now generalize to multiple hash functions - why?

Similarity for Signatures

- We know: **$Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$**
- Now generalize to multiple hash functions - why?
  - Permuting rows is expensive for large number of rows

Similarity for Signatures

- We know: **$Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$**
- Now generalize to multiple hash functions - why?
  - Permuting rows is expensive for large number of rows
  - Instead we want to simulate the effect of a random permutation using hash functions

Similarity for Signatures

- We know: **$Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$**
- Now generalize to multiple hash functions - why?
  - Permuting rows is expensive for large number of rows
  - Instead we want to simulate the effect of a random permutation using hash functions
- **The *similarity of two signatures* is the fraction of the hash functions in which they agree**

Similarity for Signatures

- We know: **$Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$**
- Now generalize to multiple hash functions - why?
  - Permuting rows is expensive for large number of rows
  - Instead we want to simulate the effect of a random permutation using hash functions
- **The *similarity of two signatures* is the fraction of the hash functions in which they agree**

Similarity for Signatures

- We know: **$Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$**
- Now generalize to multiple hash functions - why?
  - Permuting rows is expensive for large number of rows
  - Instead we want to simulate the effect of a random permutation using hash functions
- **The *similarity of two signatures* is the fraction of the hash functions in which they agree**
- **Note:** Because of the Min-Hash property, the similarity of columns is the same as the expected similarity of their signatures

Min-Hashing Example

Input matrix (Shingles x Documents)

2	4	3	1	0	1	0
3	2	4	1	0	0	1
7	1	7	0	1	0	1
6	3	2	0	1	0	1
1	6	6	0	1	0	1
5	7	1	1	0	1	0
4	5	5	1	0	1	0

Permutation π

Signature matrix *M*

2	1	2	1
2	1	4	1
1	2	1	2

Similarities:

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

Min-Hash Signatures

- **Pick K=100 random permutations of the rows**
- 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*

$$sig(C)[i] = \min(\pi_i(C))$$

- **Note:** The sketch (signature) of document *C* is small **~100 bytes!**
- **We achieved our goal! We "compressed" long bit vectors into short signatures**

Min-Hash Signatures Example

Row	S_1	S_2	S_3	S_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

Init

Min-Hash Signatures Example

Row	S_1	S_2	S_3	S_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

Init

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	1
h_2	1	∞	∞	1

Row 0

Min-Hash Signatures Example

Row	S_1	S_2	S_3	S_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

Init

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	1
h_2	1	∞	∞	1

Row 0

	S_1	S_2	S_3	S_4
h_1	1	∞	2	1
h_2	1	∞	4	1

Row 1

Min-Hash Signatures Example

Row	S_1	S_2	S_3	S_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

Init

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	1
h_2	1	∞	∞	1

Row 0

	S_1	S_2	S_3	S_4
h_1	1	∞	2	1
h_2	1	∞	4	1

Row 1

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	1	2	4	1

Row 2

Min-Hash Signatures Example

Row	S_1	S_2	S_3	S_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

Init

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	1
h_2	1	∞	∞	1

Row 0

	S_1	S_2	S_3	S_4
h_1	1	∞	2	1
h_2	1	∞	4	1

Row 1

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	1	2	4	1

Row 2

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	0	2	0	0

Row 3

Min-Hash Signatures Example

Row	S_1	S_2	S_3	S_4	$x+1 \mod 5$	$3x+1 \mod 5$
0	1	0	0	1	1	1
1	0	0	1	0	2	4
2	0	1	0	1	3	2
3	1	0	1	1	4	0
4	0	0	1	0	0	3

	S_1	S_2	S_3	S_4
h_1	∞	∞	∞	∞
h_2	∞	∞	∞	∞

Init

	S_1	S_2	S_3	S_4
h_1	1	∞	∞	1
h_2	1	∞	∞	1

Row 0

	S_1	S_2	S_3	S_4
h_1	1	∞	2	1
h_2	1	∞	4	1

Row 1

	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	1	2	4	1

Row 2

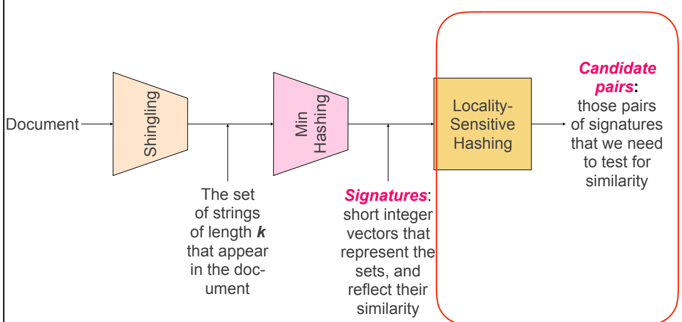
	S_1	S_2	S_3	S_4
h_1	1	3	2	1
h_2	0	2	0	0

Row 3

	S_1	S_2	S_3	S_4
h_1	1	3	0	1
h_2	0	2	0	0

Row 4

The Big Picture



LSH: First Cut

2	1	4	1
1	2	1	2
2	1	2	1

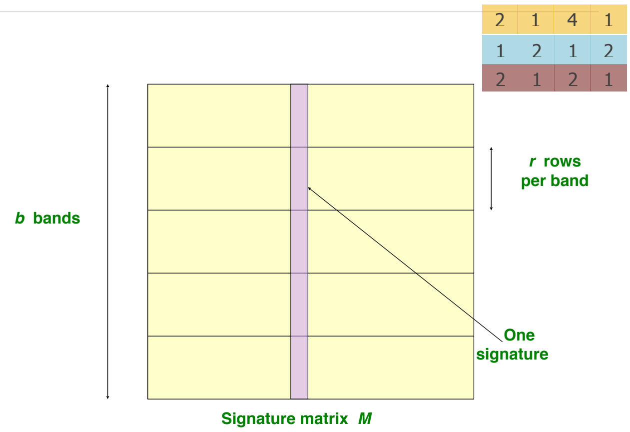
- **Goal:** Find documents with Jaccard similarity at least *s* (for some similarity threshold, e.g., *s*=0.8)
- **LSH – General idea:** Use a function *f(x,y)* that tells whether *x* and *y* is a ***candidate pair:*** a pair of elements whose similarity must be evaluated
- **For Min-Hash matrices:**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

Candidates from Min-Hash

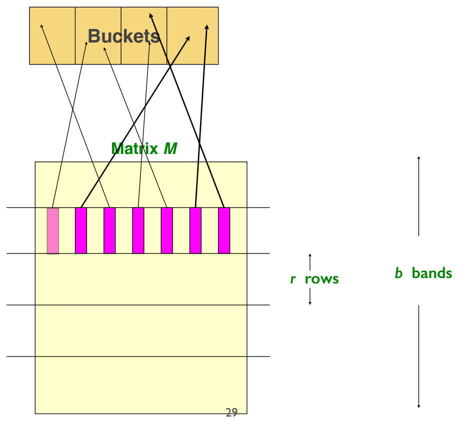
2	1	4	1
1	2	1	2
2	1	2	1

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

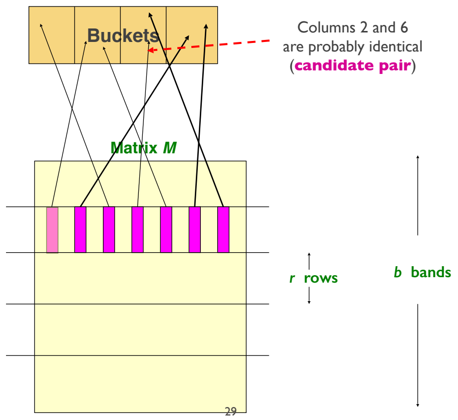
Partition *M* into *b* Bands



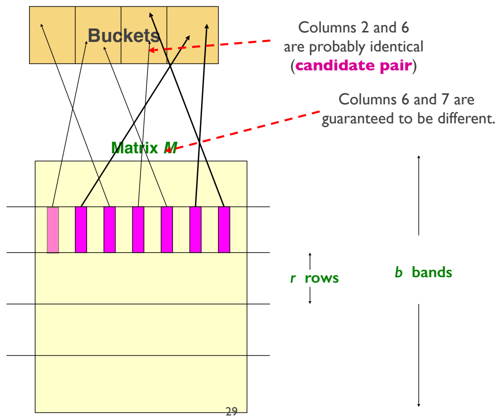
Hashing Bands



Hashing Bands



Hashing Bands



Partition M into Bands

- 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 non-similar pairs

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

b bands, *r* rows/band

- Columns C_1 and C_2 have similarity ***s***
- Pick any band (***r*** rows)
  - Prob. that all rows in band equal = s^r
  - Prob. that some row in band unequal = $1 - s^r$
- Prob. that no band identical = $(1 - s^r)^b$
- Prob. that at least one band is identical = $1 - (1 - s^r)^b$

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

C_1, C_2 are 80% Similar

- **Find pairs of ≥ *s*=0.8 similarity, set b=20, r=5**
- **Assume:** sim(C_1, C_2) = 0.8
  - Since sim(C_1, C_2) ≥ 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)

C_1, C_2 are 80% Similar

- **Find pairs of ≥ *s*=0.8 similarity, set b=20, r=5**
- **Assume:** sim(C_1, C_2) = 0.8
  - Since sim(C_1, C_2) ≥ 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:** $(0.8)^5 = 0.328$

C_1, C_2 are 80% Similar

- **Find pairs of ≥ *s*=0.8 similarity, set b=20, r=5**
- **Assume:** sim(C_1, C_2) = 0.8
  - Since sim(C_1, C_2) ≥ 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: $(1-0.328)^{20} = 0.00035$
  - i.e., about 1/3000th of the 80%-similar column pairs are **false negatives** (we miss them)
  - **We would find $1-(1-0.328)^{20}$ = 99.965% pairs of truly similar documents**

C_1, C_2 are 30% Similar

- **Find pairs of ≥ *s*=0.8 similarity, set b=20, r=5**
- **Assume:** sim(C_1, C_2) = 0.3
  - Since sim(C_1, C_2) < s we want C_1, C_2 to hash to **NO common buckets** (all bands should be different)

C_1, C_2 are 30% Similar

- **Find pairs of ≥ *s*=0.8 similarity, set b=20, r=5**
- **Assume:** sim(C_1, C_2) = 0.3
  - Since sim(C_1, C_2) < s we want C_1, C_2 to hash to **NO common buckets** (all bands should be different)
- **Probability C_1, C_2 identical in one particular band:** $(0.3)^5 = 0.00243$

C_1, C_2 are 30% Similar

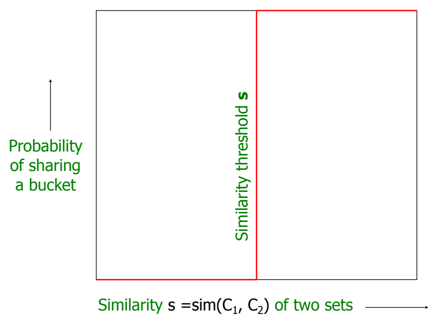
- **Find pairs of ≥ *s*=0.8 similarity, set b=20, r=5**
- **Assume:** sim(C_1, C_2) = 0.3
  - Since sim(C_1, C_2) < s we want C_1, C_2 to hash to **NO common buckets** (all bands should be different)
- **Probability C_1, C_2 identical in one particular band:** $(0.3)^5 = 0.00243$
- Probability C_1, C_2 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

LSH Involves a Tradeoff

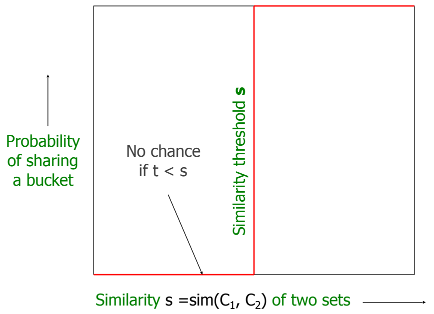
- **Pick:**
  - The number of Min-Hashes (rows of *M*)
  - The number of bands *b*, and
  - The number of rows *r* per band

  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

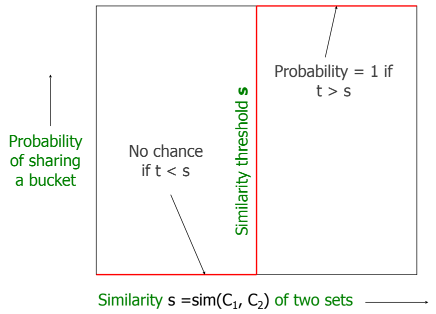
Analysis of LSH – What We Want



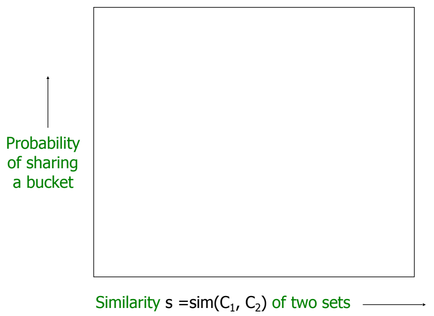
Analysis of LSH – What We Want



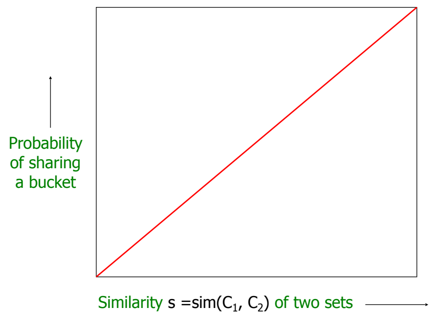
Analysis of LSH – What We Want



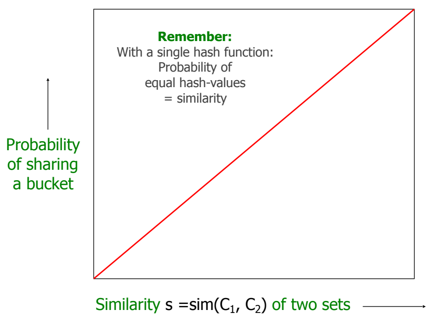
What One Band of One Row Gives You



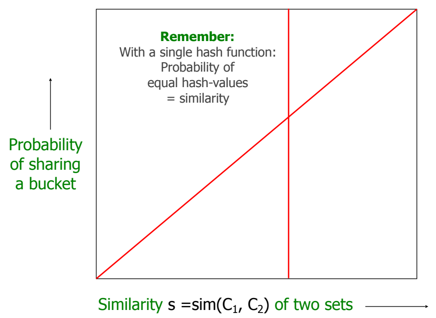
What One Band of One Row Gives You



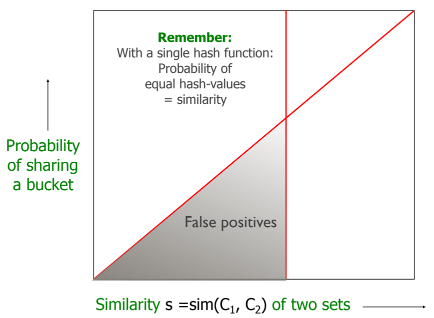
What One Band of One Row Gives You



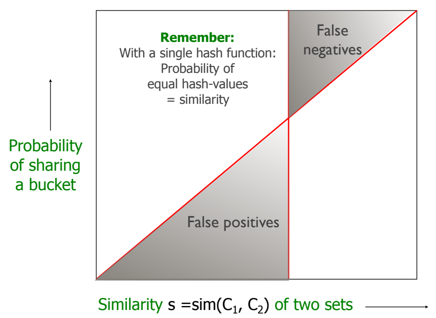
What One Band of One Row Gives You



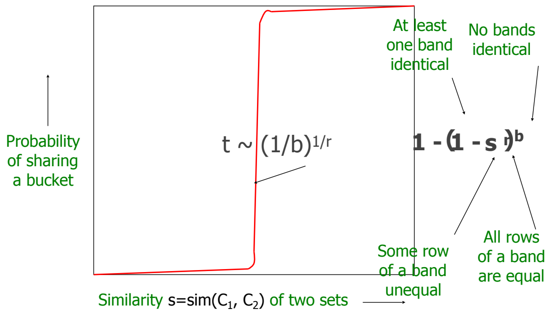
What One Band of One Row Gives You



What One Band of One Row Gives You



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



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

LSH Summary

- Tune ***M, b, r*** 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 documents

References

For LSH refer to the Mining of Massive Datasets Chapter 3 http://infolab.stanford.edu/~ullman/mmds/book.pdf

LSH slides are borrowed from http://i.stanford.edu/~ullman/cs246slides/LSH-1.pdf