Finding Similar Sets

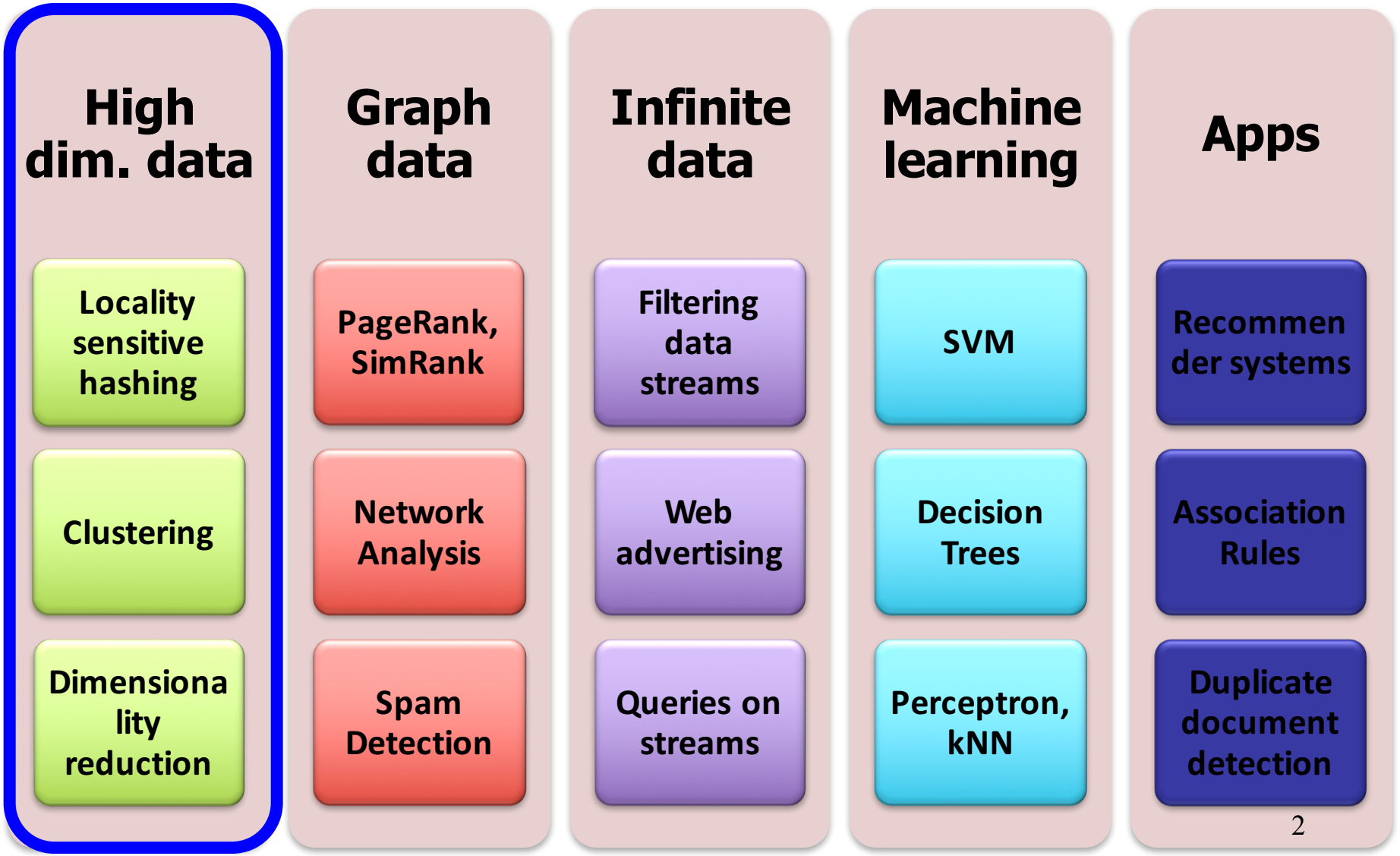
Applications

Shingling

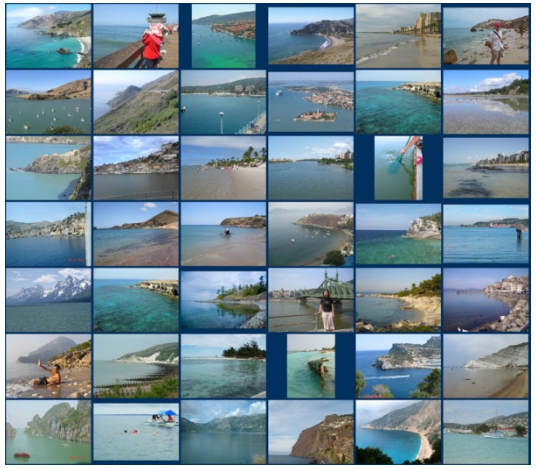
Minhashing

Locality-Sensitive Hashing

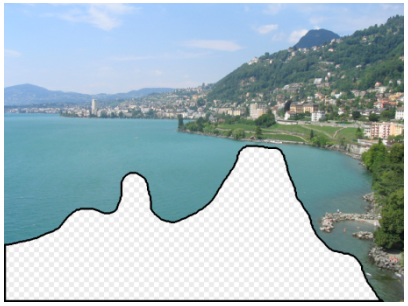
New thread: High dim. data



Scene Completion Problem



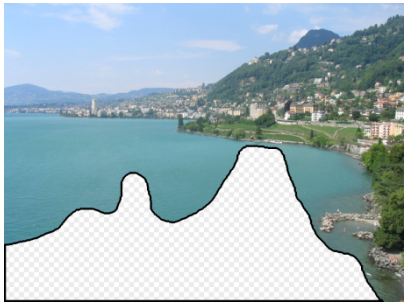
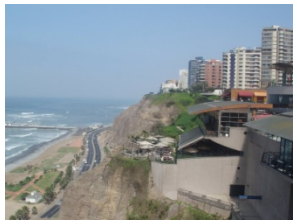
Scene Completion Problem

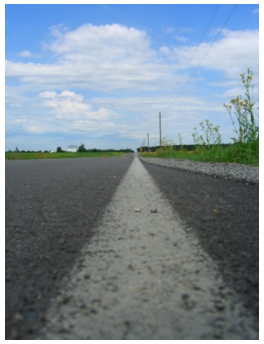


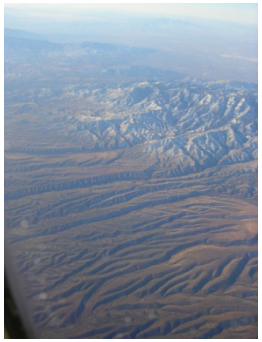
Scene Completion Problem





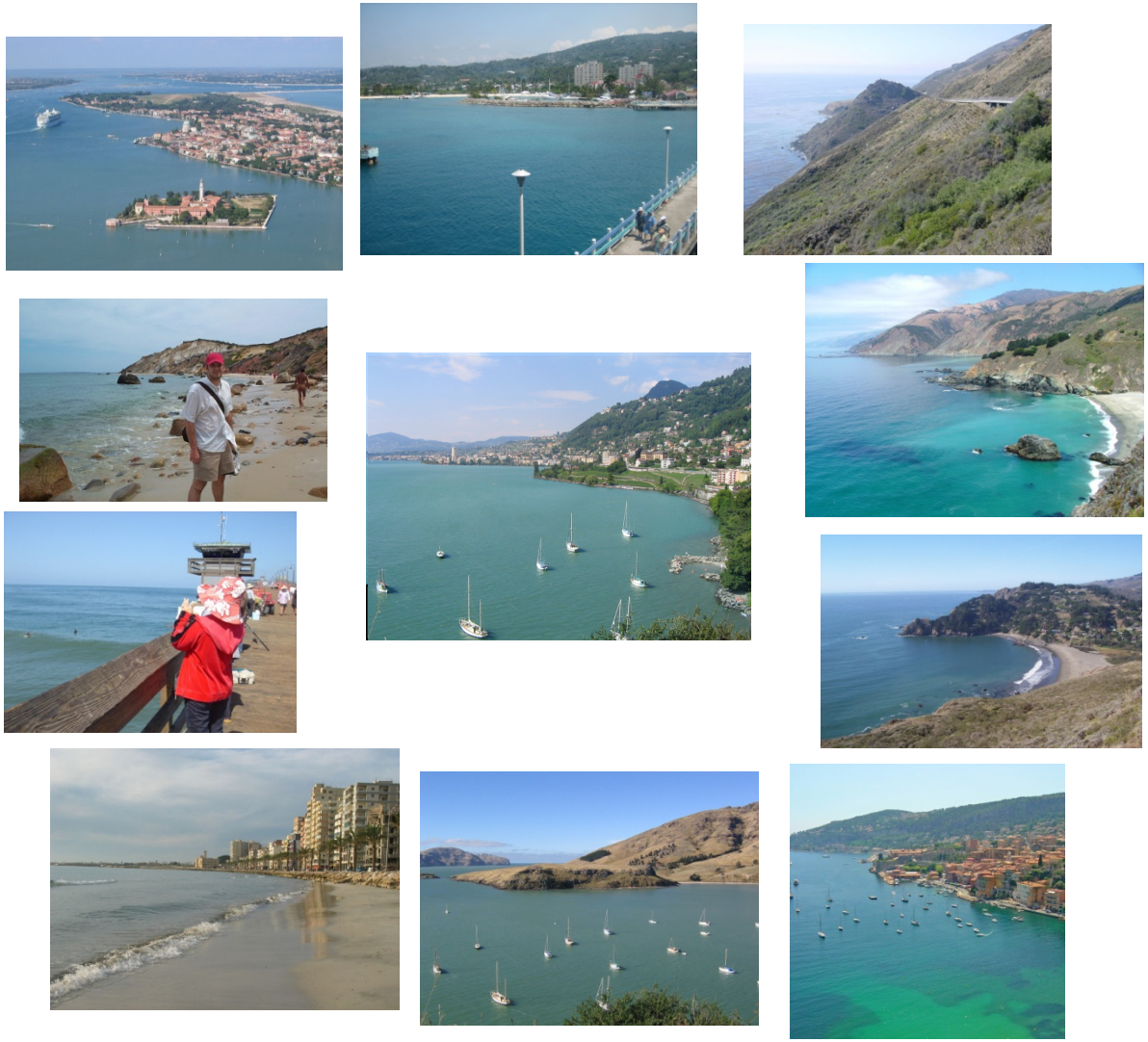





10 nearest neighbors from a collection of 20,000 images

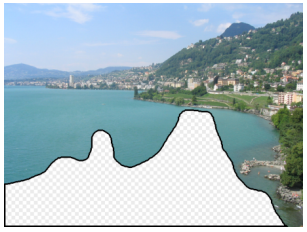
Scene Completion Problem



10 nearest neighbors from a collection of 2 million images

A Common Metaphor

- **Many problems can be expressed as finding "similar" sets:**
  - **Find near-neighbors in high-dimensional space**
- **Examples:**
  - **Pages with similar words**
    - For duplicate detection, classification by topic
  - **Movie Rating, NetFlix users with similar tastes in movies**
    - For recommendation systems
  - **Customers who purchased similar products**
    - Products with similar customer sets
  - **Images with similar features**
    - Users who visited similar websites.



Problem for Today's Lecture

- **Given: High dimensional data points $x_1, x_2, \ldots$**
  - **For example:** Image is a long vector of pixel colors

$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow [1\ 2\ 1\ 0\ 2\ 1\ 0\ 1\ 0]$$

- **And some distance function $d(x_1, x_2)$**
  - Which quantifies the "distance" between x_1 and x_2
- **Goal:** Find **all pairs of data points** (x_i, x_j) that are within some distance threshold $d(x_i, x_j) \leq s$
- **Note:** Naïve solution would take $O(N^2)$ ☹
  - where N is the number of data points
  - $O(N^2)$ represents an algorithm whose performance is directly proportional to the **square of the size** of the **input data** set
- **MAGIC: This can be done in $O(N)$!! How?**

Finding Similar Items

Finding Similar Documents

- Given a body of **documents**, e.g., the Web, find **pairs of documents** with a lot of **text in common**, such as:
  - **Mirror sites**, or approximate mirrors,
    - **Application:** Don't want to show both in a search.
  - **Plagiarism**, including large quotations
  - **Similar news articles** at many news sites,
    - **Application:** Cluster articles by "same story."

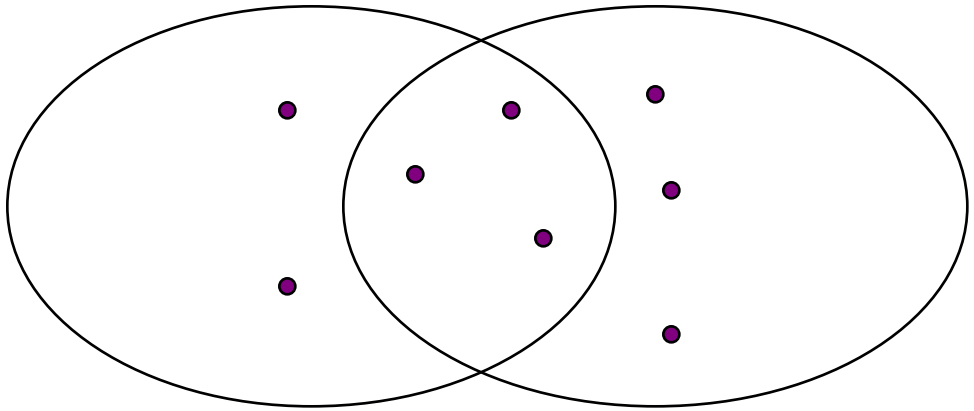
Distance Measures

- **Goal: Find near-neighbors in high-dimensional space**
  - We formally define "near neighbors" as points that are a "small distance" apart
- For each application, we first need to define what "**distance**" means
- **Today: Jaccard distance/similarity.**

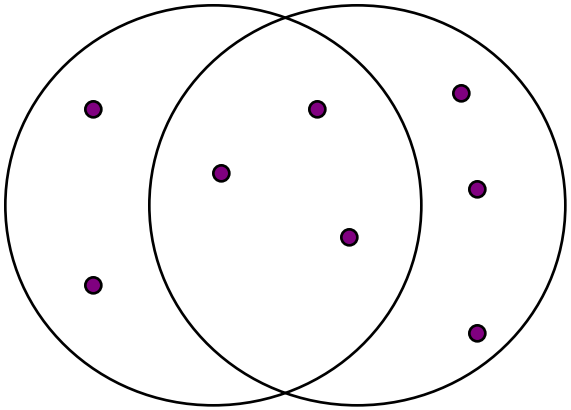
Jaccard Similarity of Sets

- The *Jaccard similarity* of two sets is the **size of their intersection** divided by the **size of their union**

$$Sim\ (C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$$



Example: Jaccard Similarity



3 in intersection
8 in union

Jaccard similarity = 3/8

- **Jaccard distance** = 1 – Jaccard Similarity or 5/8 in this example

Application: Collaborative Filtering

- Online purchase
  - Recommend items from other buyers
  - ⇒Need to find similar buyers or items
  - ⇒Buyer-to-item or item-to-item recommendation
- Buyer = a set of items he/she purchased
  - Similar buyers: users who bought many common items
- Item = a set of buyers who purchased it
  - Similar items: items having many common buyers.

Collaborative Filtering @Amazon

- Item-to-item recommendation

Customers Who Bought This Item Also Bought

Page 1 of 19

Photive iPad Air Smart Case. Lightweight Smart Cover Case for the New iPad Air with Built in...
848
$19.95 Prime

iPad Air Case, SUPCASE Heavy Duty Beetle Defense Series Full-body Rugged Hybrid Protective Case...
2,193
$19.99 Prime

Tech Armor Apple iPad Air 2 / iPad Air (first generation) High Defintion (HD) Clear Screen...
4,686
#1 Best Seller in Tablet Screen Protectors
$9.95 Prime

Fintie iPad Air Case - SmartShell Case for Apple iPad Air (iPad 5) 2013 Model, Ultra Slim...
2,471
$9.99 Prime

Fintie Apple iPad Air Case - 360 Degree Rotating Stand Case Cover with Auto Sleep / Wake Feature for...
1,795
$14.99 Prime

Finding Similar Buyers

Who is most similar to B1?

B1	{A, B}
B2	{A, B, D}
B3	{A, C, D}

Buyer-to-item recommendation

Jaccard(B1, B2) = 2/3 ⇒ B2 most similar to B1 ⇒ **B2 also bought D**

Jaccard(B1, B3) = 1/4

⇒ Recommend **D to B1**

Who is most similar to B2?

Jaccard(B2, B1) = Jaccard(B1, B2)

Jaccard(B2, B3) = 1/2

Who is most similar to B3?

Jaccard(B3, B1) = Jaccard(B1, B3)

Jaccard(B3, B2) = Jaccard(B2, B3)

Finding Similar Items

B1	{A, B}
B2	{A, B, D}
B3	{A, C, D}

Buyer as a set

A	{B1, B2, B3}
B	{B1, B2}
C	{B3}
D	{B2, B3}

Item as a set

Finding Similar Items

Which item is most similar to A?

A	{B1, B2, B3}
B	{B1, B2}
C	{B3}
D	{B2, B3}

Which item is most similar to B?

Finding Similar Items

Which item is most similar to A?

A	{B1, B2, B3}
B	{B1, B2}
C	{B3}
D	{B2, B3}

Jaccard(A, B) = 2/3
Jaccard(A, C) = 1/3
Jaccard(A, D) = 2/3

→ A is most similar to B and D

Which item is most similar to B?

Finding Similar Items

Which item is most similar to A?

A	{B1, B2, B3}
B	{B1, B2}
C	{B3}
D	{B2, B3}

Jaccard(A, B) = 2/3
Jaccard(A, C) = 1/3
Jaccard(A, D) = 2/3

→ A is most similar to B and D

Which item is most similar to B?

Jaccard(B, A) = Jaccard(A, B) = 2/3
Jaccard(B, C) = 0
Jaccard(B, D) = 1/3

→ B is most similar to A

Formulated as frequent itemset problem?

- Find similar items
  - Items bought together by many users

  ⇒User = transaction

  ⇒Similar items = frequent item pair

B1	{A, B}
B2	{A, B, D}
B3	{A, C, D}

- Find similar users
  - Users that bought many common items
  - Item = transaction

  => Find frequent user pairs

A	{B1, B2, B3}
B	{B1, B2}
C	{B3}
D	{B2, B3}

Finding Similar Items

Which item is most similar to A?

A	{B1, B2, B3}
B	{B1, B2}
C	{B3}
D	{B2, B3}

Jaccard(A, B) = 2/3
Jaccard(A, C) = 1/3
Jaccard(A, D) = 2/3

→ A is most similar to B and D

Which item is most similar to B?

Jaccard(B, A) = Jaccard(A, B) = 2/3
Jaccard(B, C) = 0
Jaccard(B, D) = 1/3

→ B is most similar to A

B1	{A, B}
B2	{A, B, D}
B3	{A, C, D}

Item-to-item recommendation

B1 has bought A, B (or put them in basket)
A is most similar to B and D
B is most similar to A

→ **Recommend: D to B1**

Finding Frequent Item Pairs

B1	{A, B}
B2	{A, B, D}
B3	{A, C, D}

Sup(A) = 3
Sup(B) = 2
Sup(C) = 1
Sup(D) = 2

min_sup = 2 → prune

Apriori algorithm

Finding Frequent Item Pairs

B1	{A, B}
B2	{A, B, D}
B3	{A, C, D}

Sup(A) = 3
Sup(B) = 2
Sup(C) = 1
Sup(D) = 2

min_sup = 2

prune

Sup(A) = 3
Sup(B) = 2
Sup(D) = 2

Candidate 2-itemsets

(A,B)
(A,D)
(B,D)

Apriori algorithm

Finding Frequent Item Pairs

B1	{A, B}
B2	{A, B, D}
B3	{A, C, D}

Sup(A, B) = 2
Sup(A, C) = 1
Sup(A, D) = 2
Sup(B, C) = 0
Sup(B, D) = 1
Sup(C, D) = 1

min_sup = 2 →

(A, B)
(A, D)

Sup(A) = 3
Sup(B) = 2
Sup(C) = 1
Sup(D) = 2

min_sup = 2 → prune

Sup(A) = 3
Sup(B) = 2
Sup(D) = 2

Candidate 2-itemsets →

(A,B)
(A,D)
(B,D)

Apriori algorithm

Finding Frequent User Pairs

A	{B1, B2, B3}
B	{B1, B2}
C	{B3}
D	{B2, B3}

Sup(B1) = 2
Sup(B2) = 3
Sup(B3) = 3

min_sup = 2 → prune

Apriori algorithm

Finding Frequent User Pairs

A	{B1, B2, B3}
B	{B1, B2}
C	{B3}
D	{B2, B3}

Sup(B1) = 2
Sup(B2) = 3
Sup(B3) = 3

min_sup = 2

prune

Sup(B1) = 2
Sup(B2) = 3
Sup(B3) = 3

Candidate 2-itemsets

(B1,B2)
(B1,B3)
(B2,B3)

Apriori algorithm

Finding Frequent User Pairs

A	{B1, B2, B3}
B	{B1, B2}
C	{B3}
D	{B2, B3}

Sup(B1, B2) = 2
Sup(B1, B3) = 1
Sup(B2, B3) = 2

min_sup = 2

(B1, B2)
(B2, B3)

min_sup = 2

Candidate 2-itemsets

(B1,B2)
(B1,B3)
(B2,B3)

Sup(B1) = 2
Sup(B2) = 3
Sup(B3) = 3

min_sup = 2

prune

Sup(B1) = 2
Sup(B2) = 3
Sup(B3) = 3

Apriori algorithm

So why new solution?

- Just use Apriori to find
  - Frequent item pairs => similar items

B1	{A, B}
B2	{A, B, D}
B3	{A, C, D}

  - Frequent user pairs => similar users

A	{B1, B2, B3}
B	{B1, B2}
C	{B3}
D	{B2, B3}

Potential Problems

- Are buyer B1 and B2 still similar?
- What would Apriori say?

B1	{A, B}
B2	{A, B, D}
B3	{A, C, D}
B4	{E, F}
B5	{E, F, G}
B6	{X, Y, Z}
B7	{X, W}
B8	{S, T, O}
B9	{S, T}
…	…

Potential Problems

- Are buyer B1 and B2 still similar?

- What would Apriori say?
  - Form item->buyers basket
  - A large number of items
  - B1 and B2 only appear together in two baskets
  - Hence very low support
  - But they are still similar.

B1	{A, B}
B2	{A, B, D}
B3	{A, C, D}
B4	{E, F}
B5	{E, F, G}
B6	{X, Y, Z}
B7	{X, W}
B8	{S, T, O}
B9	{S, T}
…	…

Application: Collaborative Filtering

- Recommend movies
  - Recommend similar movies or
  - Movies from similar users

- User = a set of movies he/she has watched

- Movie = a set of users who has watched it

What if we consider ratings too?

- User A watched movie HP1 (Harry Potter 1)
  - And rated it 4
- May need different similarity function/solution

⇒Recommendation systems (Chapter 9)

	HP1	HP2	HP3	TW	SW1	SW2	SW3
A	4			5	1		
B	5	5	4				
C				2	4	5	
D		3					3

Task: Finding Similar Documents

- **Goal: Given a large number (in the millions or billions) of documents, find "near duplicate" pairs**
- **Applications:**
  - Mirror websites, or approximate mirrors
    - Don't want to show both in search results
  - Similar news articles at many news sites
    - Cluster articles by "same story"
- **Problems:**
  - Many **small pieces** of one document can appear out of order in another
  - **Too many** documents **to compare** all pairs
  - Documents are so large or so many that they **cannot fit in main memory.**

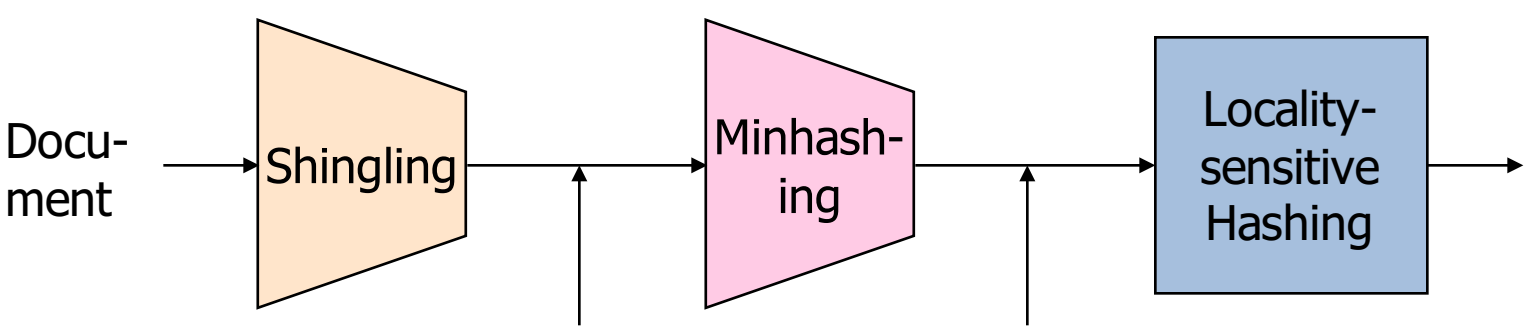
3 Essential Steps for Finding Similar Docs

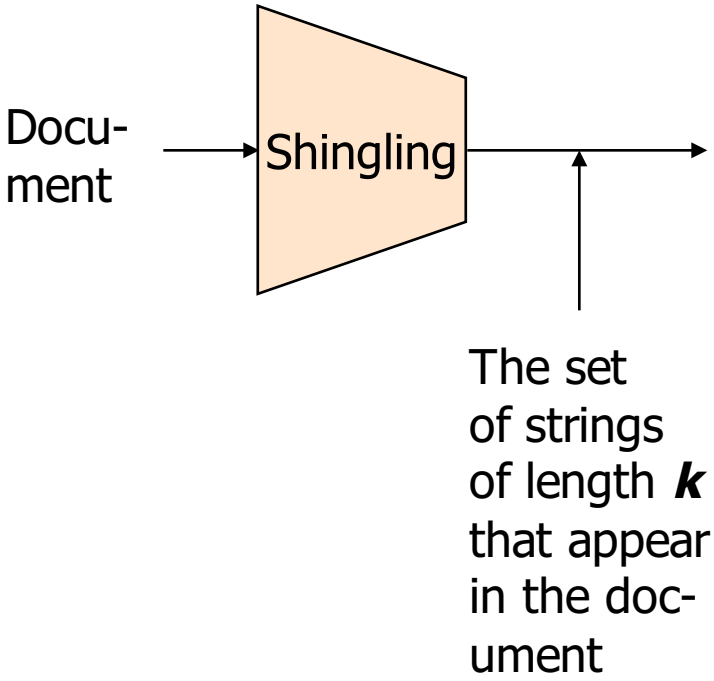
1. ***Shingling:*** Convert documents **to sets**

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





Shingling

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

Documents as High-Dimensional 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}.

Shingles

- Web page as a string of characters
- Shingle = subsequence of k-characters

- Web page = abcdabd, k = 2
- 2-shingles
  - **ab, bc, cd, da, bd**

- Max # of k-shingles for a page of n characters?
  - **7 characters, 6 max 2-shingles, two identical, 5 unique**
  - **Max = 7 – 2 + 1 = 6**
  - **N – k + 1 ~ n**

White Spaces

- Better not omit them
- Could turn multiple into one

- D1: “scored a touch down” => “scored a touch down”
- D2: “touchdown at last”

- D1 and D2 have a common 9-shingle if space omitted.

Shingle Size

- Too small
  - Many documents will falsely become similar
- Too big
  - Might miss truly similar documents.

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 (eg. Email, Tweet)
  - ***k*** = 10 is better for long documents
  - news articles, blog posts (in between)

- May want to **compress long shingles**.

Compressing Shingles

- To **compress long shingles**, we can **hash** them to (say) 4 bytes
  - Called ***tokens***
- **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}.

Why is compression needed?

- How many k-shingles?
  - Rule of thumb: imagine **20 characters** in alphabet
    - **26 characters + whitespace = 27**
    - **since "z,q,x" used rarely**
    - A **k-shingle** is a sequence of k consecutive words (or k-gram)
  - Estimate of number of k-shingles is 20^k
  - 4-shingles: 20^4 or 160,000 or $2^{17.3}$
  - 9-shingles: 20^9 or 512,000,000,000 or $\mathbf{2^{39}}$
- Assume we use **4 bytes** to represent a **bucket**
- Buckets numbered in range 0 to $2^{32} - 1$
- Much smaller than possible number of **9-shingles** and represent each shingle with 4 bytes, not 9 bytes
  - Compression.

Thought Question

- Why is it better to hash 9-shingles (say) to 4 bytes than to use 4-shingles?
- Hint: How random are the 32-bit sequences that result from 4-shingling?

Why hash 9-shingles to 4 bytes rather than use 4-shingles?

- With 4-shingles, most sequences of four bytes are unlikely or impossible to find in typical documents
- Effective number of different shingles much less than $2^{32} - 1$
- With 9-shingles, 2^{39} possible shingles
  - Many more than 2^{32} buckets
- After hashing, may get any sequence of 4 bytes.

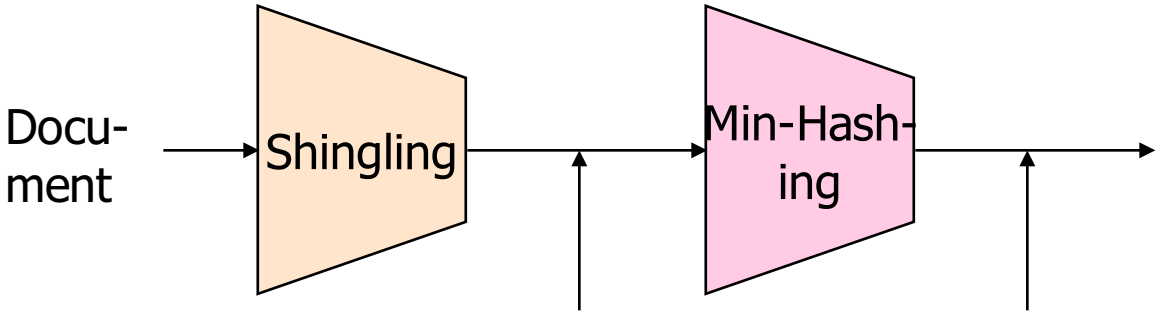
Similarity Metric for Shingles

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

Motivation for Minhash/LSH

Use k-shingles to create Signatures: short integer vectors that represent sets and reflect their similarity

- **Suppose we need to find near-duplicate documents among million documents**
- Naïvely, we would have to compute **pairwise Jaccard similarities** for **every pair of docs**
  - For N = 1 million
  - i.e, **N (N-1) /2 ≈ $5*10^{11}$** 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 set of strings of length *k* that appear in the doc-ument

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

MinHashing

Step 2: ***Minhashing:*** Convert **large sets** to **short signatures**, while **preserving similarity**

From Sets to Boolean Matrices

- Rows = elements of the universal set
- Columns = sets
  - 1 in **row *e*** and **column *S*** if and only if element *e* is a member of set *S*
  - Column similarity is the Jaccard similarity of the sets of their rows with 1: intersction/union of sets
- **Typical matrix is sparse** (many 0 values)
- May not really represent the data by a boolean matrix
- Sparse matrices are usually better represented by the list of non-zero values (e.g., triples)
  - But the matrix picture is conceptually useful.

Example 3.6

Element	S_1	S_2	S_3	S_4
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

- Universal set: {a, b, c, d, e}
- Matrix represents sets chosen from universal set
- S1 = {a, d}, S2 = {c}, S3 = {b, d, e} and S4 = {a, c, d}
- Example: **rows are products** and **columns are customers**, represented by **set of items** they bought
- **Jacquard similarity of S1, S4** ? intersection/union = 2/3

Example: Jaccard Similarity of Columns

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

Sim (C_1, C_2) = 2/5 = 0.4

Four Types of Rows

- Given columns C1 and C2, rows may be classified as:

	C1	C2
a	1	1
b	1	0
c	0	1
d	0	0

  - Also, *a* = # rows of type *a* , etc.
- Note *Sim*(C1, C2) = *a*/(*a* +*b* +*c*).

When Is Similarity Interesting?

1. When the **sets are so large** or so many that they **cannot fit** in main **memory**
2. Or, when there are **so many sets** that **comparing all pairs** of sets takes **too much time**
3. Or both.

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.

Warnings

1. Comparing **all pairs of signatures** may take **too much time**, even if not too much space
   - A job for Locality-Sensitive Hashing

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

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

Four Types of Rows

- Given columns 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

- Also, *a* = # rows of type *a* , etc.

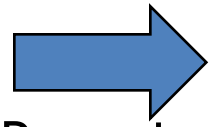
- Note *Sim* (C_1, C_2) = *a* /(*a* +*b* +*c*)
  - Jacquard similarity: intersection/union
  - a is intersection, a+b+c is union

Minhashing

1. To ***minhash*** a set represented by a column of the matrix, **pick a random permutation of the rows**
2. **Define "hash" function *h* (*C*) = the number of the first (in the permuted order) row in which column *C* has 1**
3. Use several (e.g., 100) independent hash functions to **create a signature**.

Minhashing Example (3.7)

Element	S_1	S_2	S_3	S_4
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0



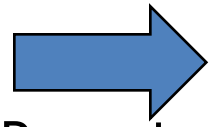
Element	S_1	S_2	S_3	S_4
b	0	0	1	0
e	0	0	1	0
a	1	0	0	1
d	1	0	1	1
c	0	1	0	1

1. To minhash a set represented by a column of the characteristic matrix, pick a **permutation of the rows**
2. The **minhash value** of any column is the number of first row, in permuted order, in which column **has a 1**
3. For set S1, first 1 appears in row a, so:

- h(S1) =
- h(S2) =
- h(S3) =
- h(S4) =

Minhashing Example (3.7)

Element	S_1	S_2	S_3	S_4
a	1	0	0	1
b	0	0	1	0
c	0	1	0	1
d	1	0	1	1
e	0	0	1	0

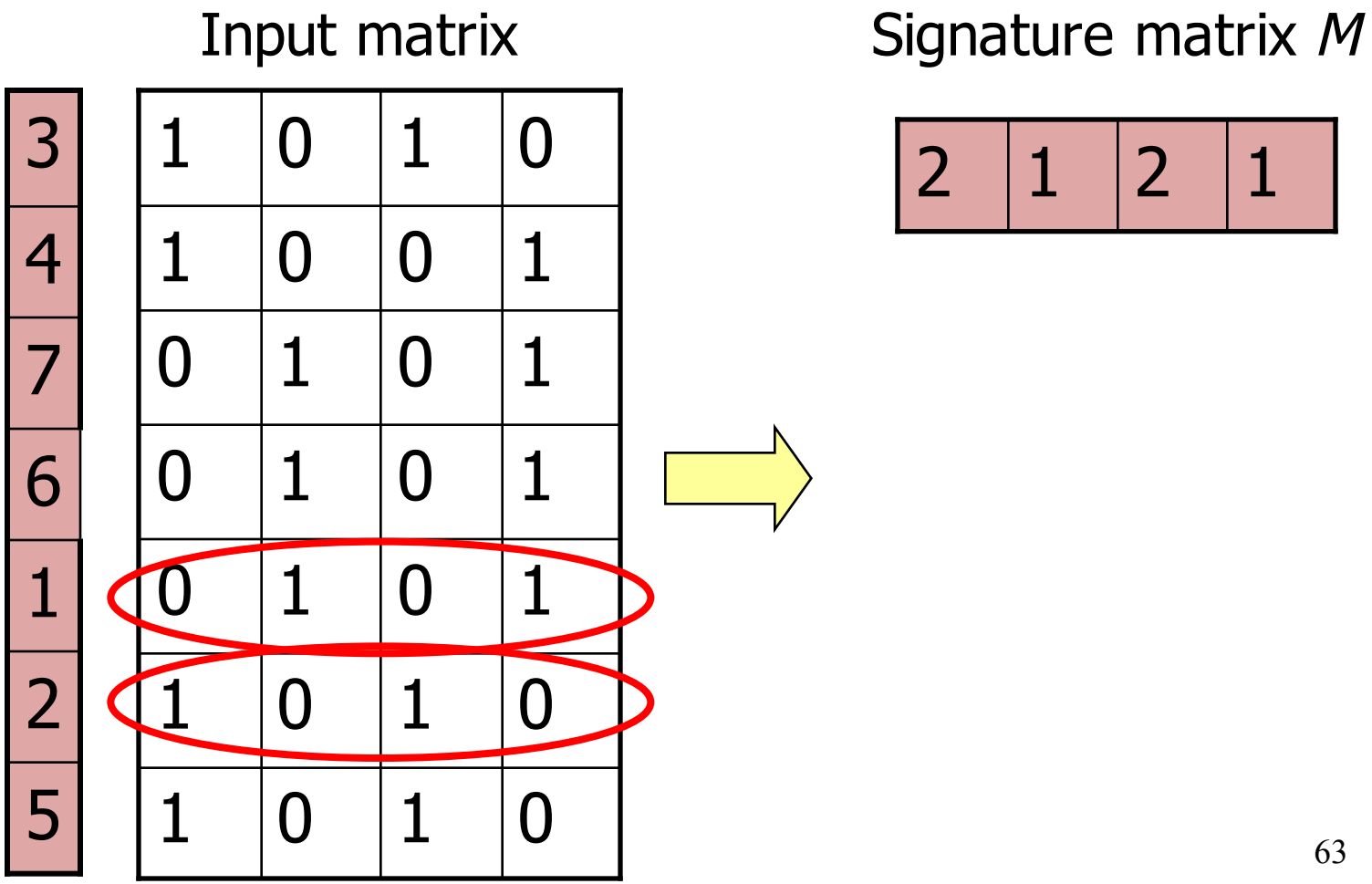


Element	S_1	S_2	S_3	S_4
b	0	0	1	0
e	0	0	1	0
a	1	0	0	1
d	1	0	1	1
c	0	1	0	1

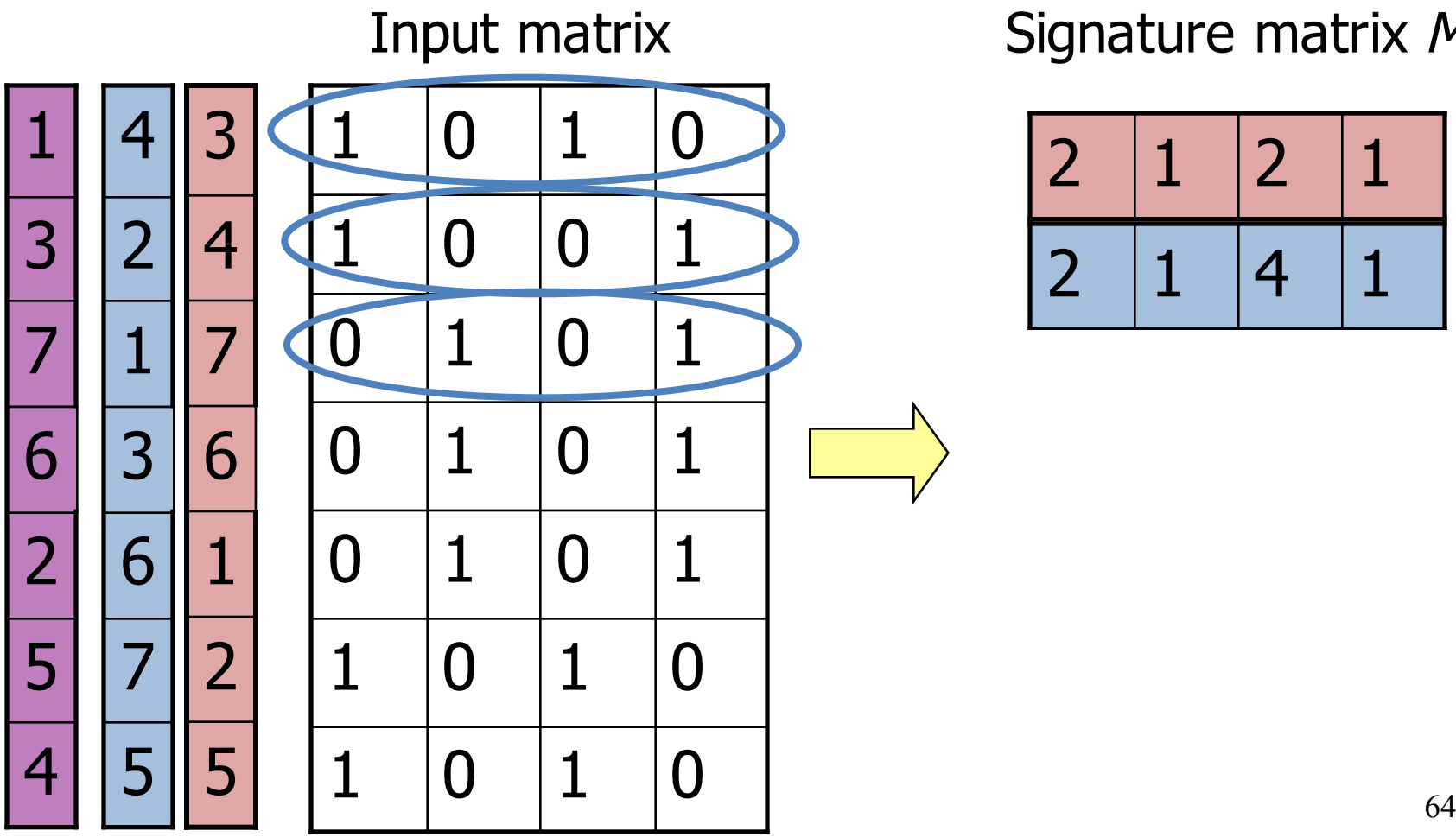
1. To minhash a set represented by a column of the characteristic matrix, pick a **permutation of the rows**
2. The **minhash value** of any column is the number of first row, in permuted order, in which column **has a 1**
3. For set S1, first 1 appears in row a, so:

- h(S1) = a
- h(S2) = c
- h(S3) = b
- h(S4) = a

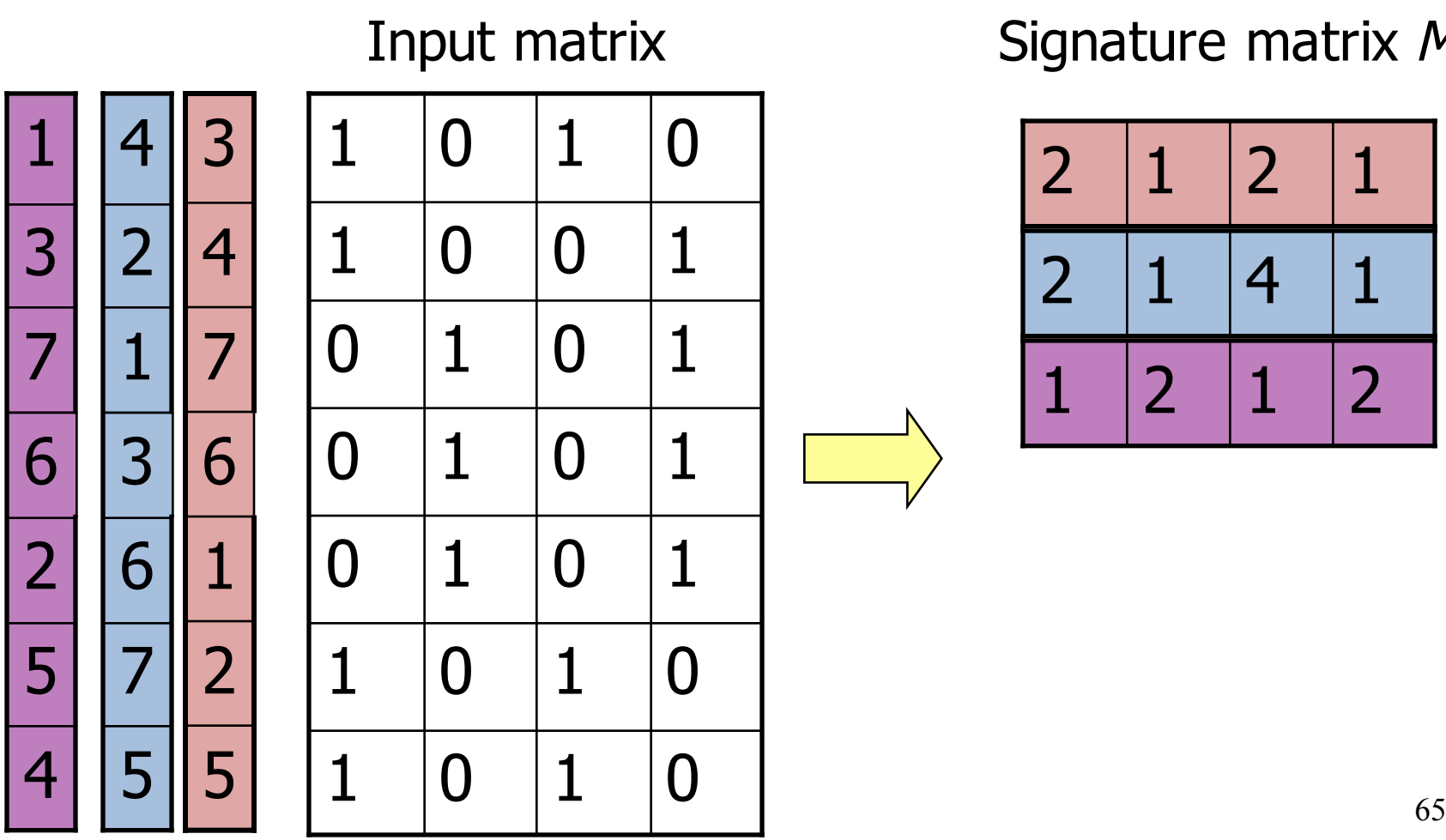
Minhashing Example



Minhashing Example



Minhashing Example



Surprising Property: Connection between Minhashing and Jaccard Similarity

- The probability that minhash function for a **random permutation of rows** produces same value for two sets equals **Jaccard similarity** of those sets
  - **Probability that $h(C_1) = h(C_2)$ is the same as $Sim(C_1, C_2)$**
- Recall four types of rows:

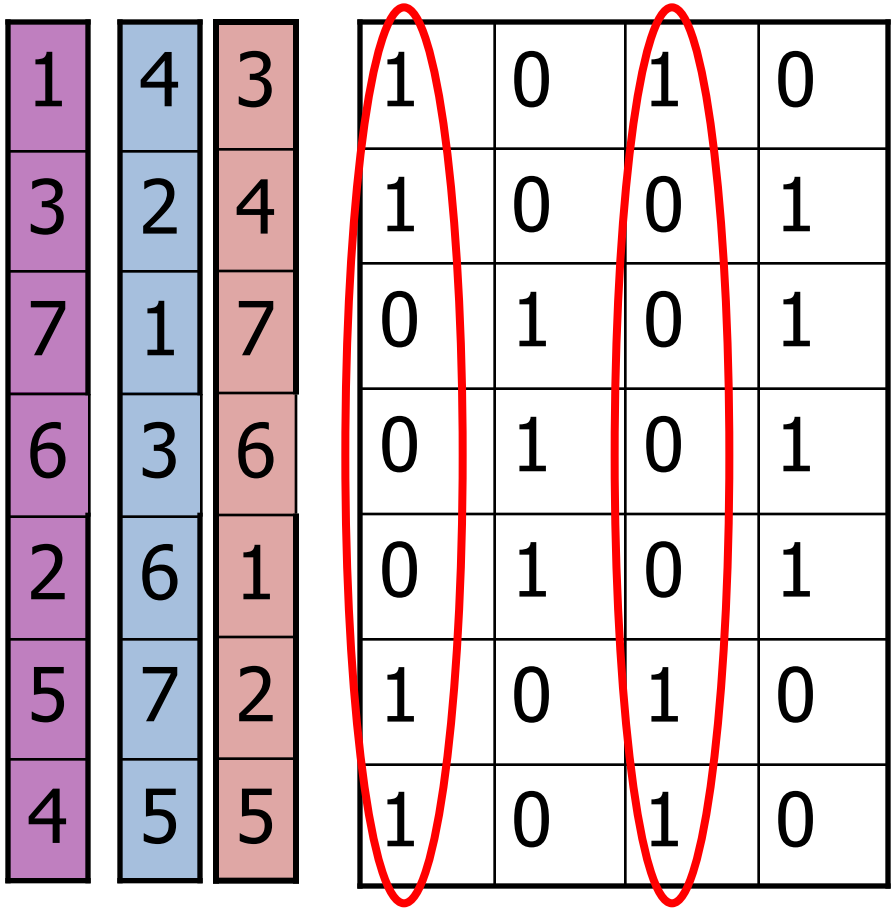
	C_1	C_2
a	1	1
b	1	0
c	0	1
d	0	0

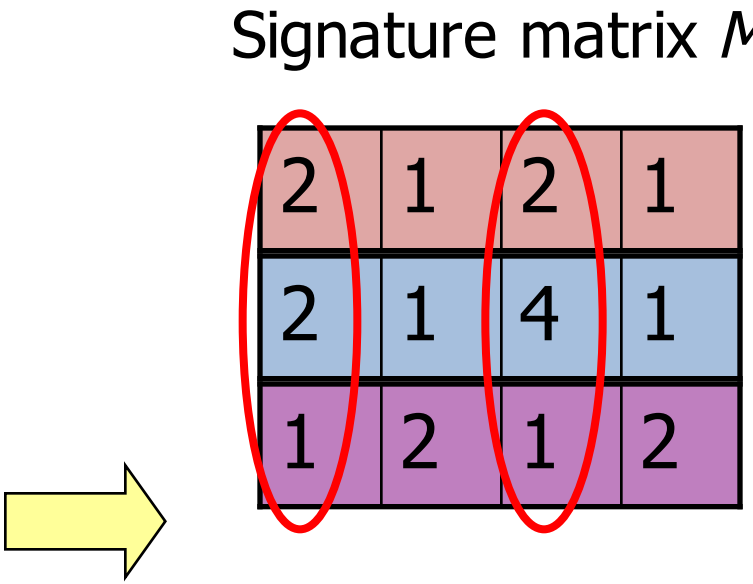
- **$Sim(C_1, C_2)$ for both Jacquard and Minhash are $a/(a+b+c)$!**
  - **Why?** Look down the permuted columns C_1 and C_2 until we see a 1
  - If it's a type-*a* row, then $h(C_1) = h(C_2)$. If a type-*b* or type-*c* row, then not. (Don't count the *type-d* rows).

Similarity for Signatures

- **Sets represented** by characteristic **matrix M**
- **To represent sets:** pick at random some number **n of permutations** of the rows of M
  - **100 permutations** or several hundred
- Call **minhash** functions determined by these permutations $h_1, h_2, \ldots, h_n$
- From **column representing set S**, construct **minhash signature for S**:
  - vector $[h_1(S), h_2(S), \ldots, h_n(S)]$, usually represented as column
- Construct a ***signature matrix***: **i^{th} column of M** replaced by **minhash signature for i^{th} column**
- **The *similarity of signatures* is the fraction of the hash functions in which they agree.**

Min Hashing – Example

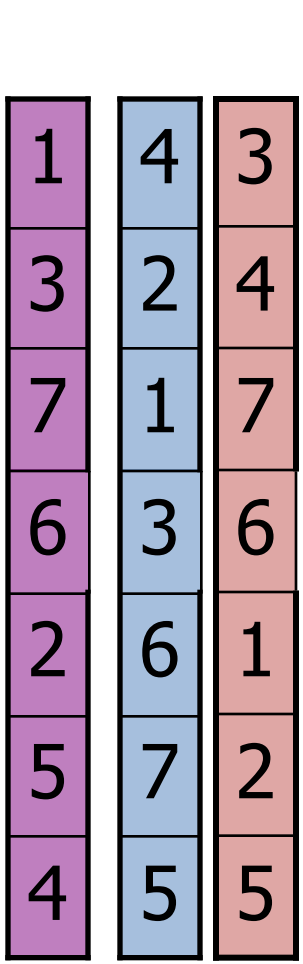




Similarities:

	1-3	2-4	1-2	3-4
	¾			
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0
	2/3			

Min Hashing – Example



Input matrix

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

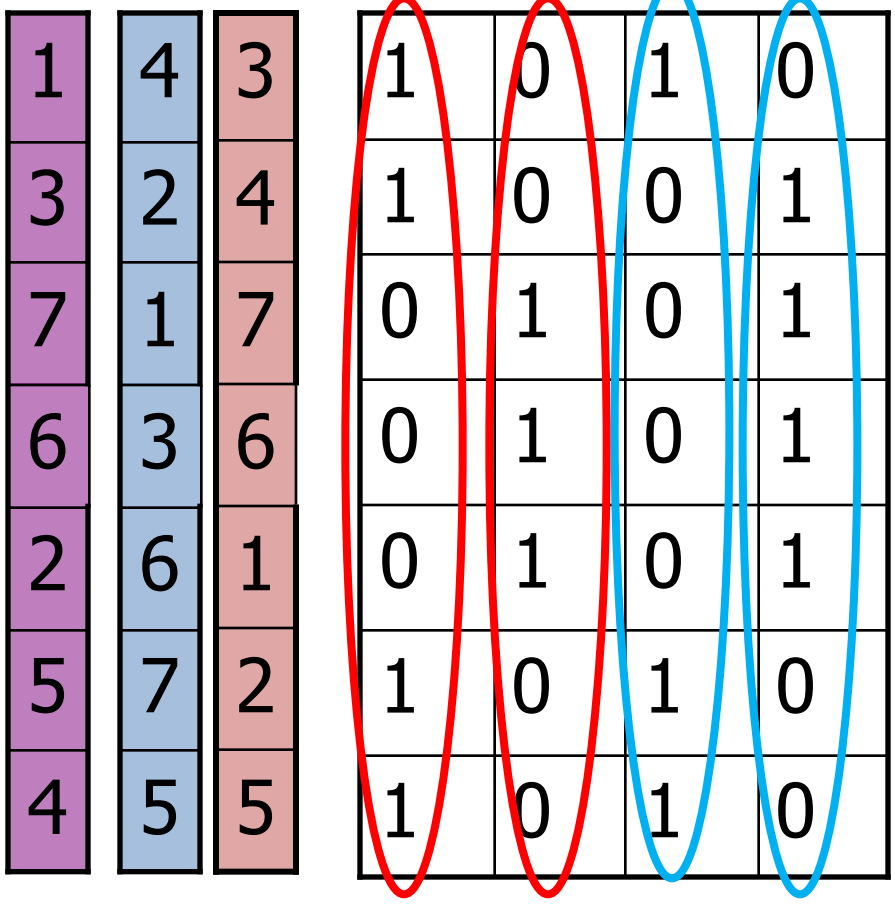
Signature matrix *M*

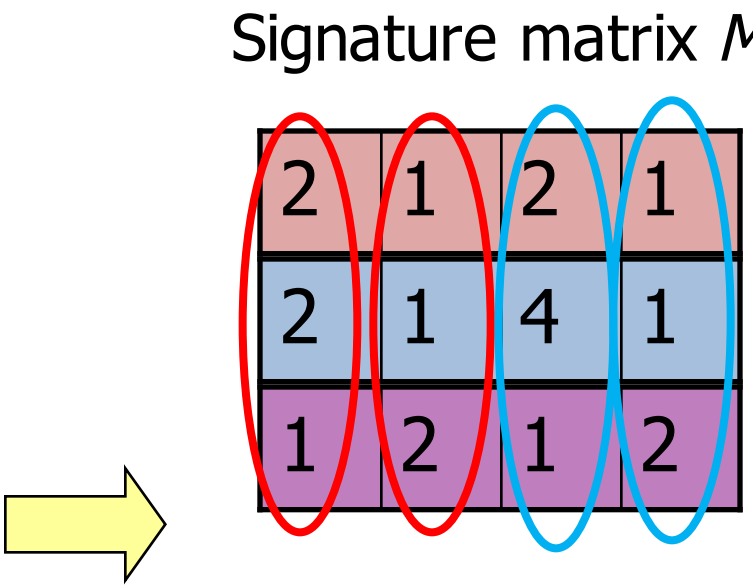
2	1	2	1
2	1	4	1
1	2	1	2

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 Hashing – Example





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

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)

- **Not feasible to permute** a large characteristic matrix explicitly
  - 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

- **Can simulate the effect of a random permutation by a random hash function**
  - **Maps row** numbers to as many buckets as there are rows
  - May have **collisions on buckets**
  - **Not important as long as number of buckets is large.**

Implementation – (2)

- A good approximation to permuting rows: pick around 100 hash functions
- For each:
  - column c (set representing a document)
  - hash function h_i
- Keep a **"slot"** in signature matrix $M(i,c)$
- **Intent: $M(i,c)$ will become the smallest value of $h_i(r)$ for which column c has 1 in row r**
  - $h_i(r)$ gives order of rows for i^{th} permuation.

Implementation – (3)

for each **row *r* do begin**

  for each hash function h_i **do**

    Compute $h_i(r)$

    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**

            $M(i, c) := h_i(r)$;

end;

Computing Minhash Signatures: Example 3.8

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

Two hash functions give permutations of rows:
h1 = x+1 mod 5, h2 = 3x +1 mod 5

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

Initial signature matrix

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

For row 0: Replace existing signature values with lower hash values for S1 and S4, since both have 1 in row

Computing Minhash Signatures: Example 3.8 (part 2)

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	1	∞	2	1
h_2	1	∞	4	1

For row 1: replace h1 and h2 values for S3, since row has a 1 and values are lower

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

For row 2: replace values for S2 since set has a 1 value. Do not replace values for S4, because existing values are lower

Computing Minhash Signatures: Example 3.8 (part 3)

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	1	3	2	1
h_2	0	2	0	0

For row 3: don't replace h1 values--all are below 4; replace h2 values with 0 for S1, S3, S4

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

For row 4: replace h1 value for S3, don't replace h2 value since current value is lower

Note: result is same as permutations to find first 1

Example

Row	C1	C2
1	1	0
2	0	1
3	1	1
4	1	0
5	0	1

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

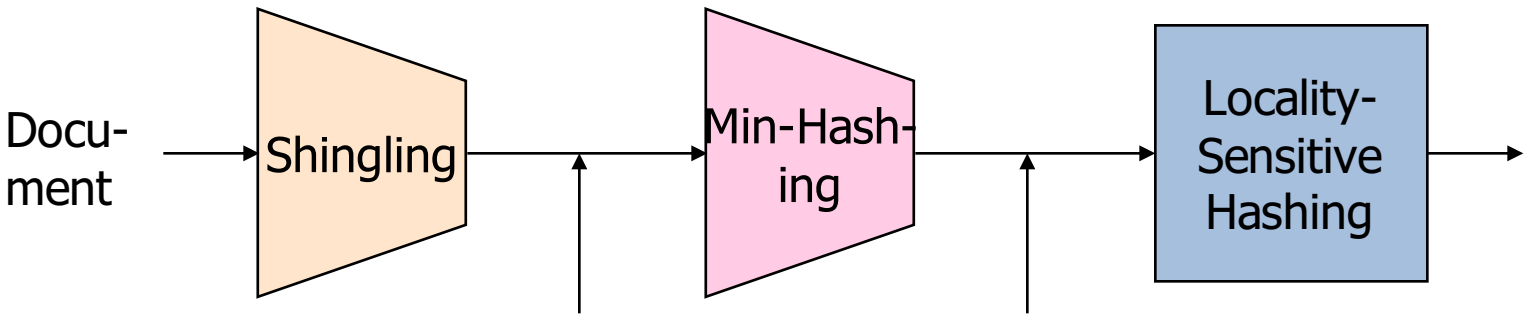
	Sig1	Sig2
$h(1) = 1$	1	-
$g(1) = 3$	3	-
$h(2) = 2$	1	2
$g(2) = 0$	3	0
$h(3) = 3$	1	2
$g(3) = 2$	2	0
$h(4) = 4$	1	2
$g(4) = 4$	2	0
$h(5) = 0$	1	0
$g(5) = 1$	2	0

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.

So far ...

- Represent a document as a set of hash values (of its k-shingles)
- Transform set of k-shingles to a set of minhash signatures
- Use Jaccard to compare two documents by comparing their signatures
- Is this method (i.e., transforming sets to signature) necessarily “better”??



Locality Sensitive Hashing

Step 3: ***Locality-Sensitive Hashing:***

Focus on pairs of signatures likely to be from similar documents