Data Mining and Analysis

Finding similar items

CSCE 676 :: Fall 2019
Texas A&M University
Department of Computer Science & Engineering
Prof. James Caverlee

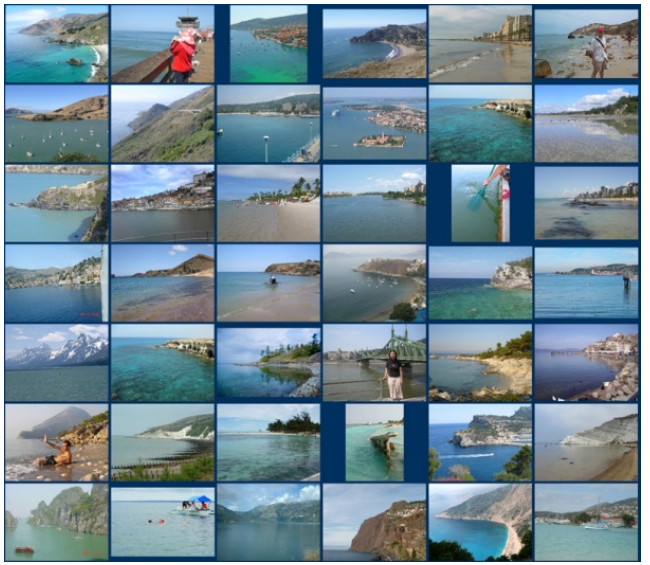
Resources

MMDS Chapter 3 + slides

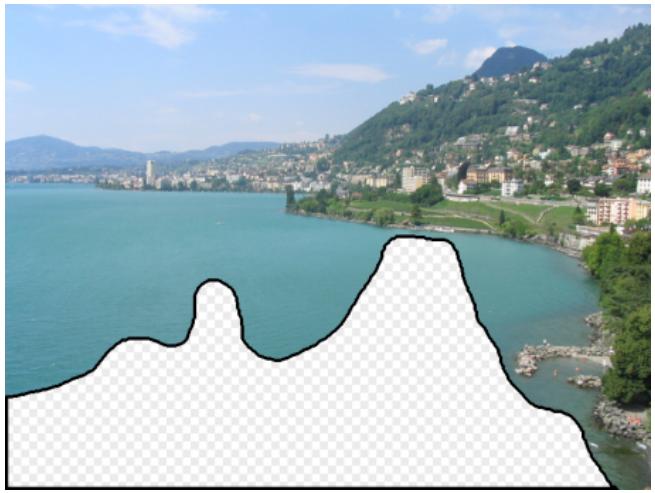
- http://i.stanford.edu/~ullman/mmds/ch3n.pdf
- http://www.mmds.org/mmds/v2.1/ch03-lsh.pdf

Carlos Castillo course on Data Mining [https://github.com/chatox/data-mining-course]

Scene Completion (SIGGRAPH 2007)



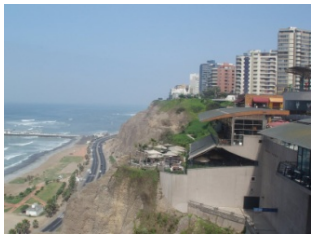
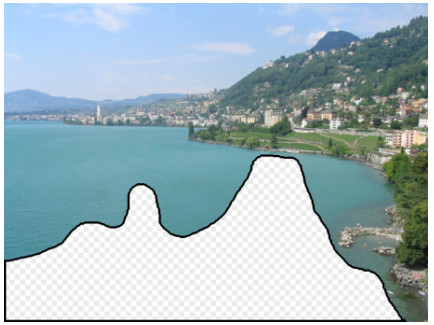
Scene Completion (SIGGRAPH 2007)

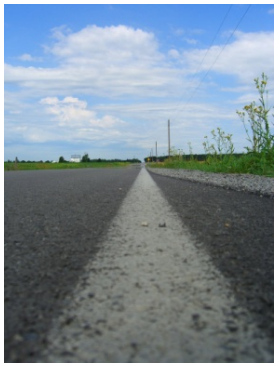


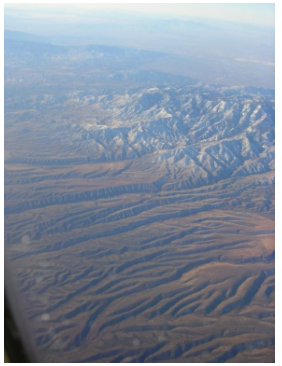
10 nearest neighbors (out of 20,000 images)







10 nearest neighbors (out of 2 million images)











Finding nearest “baskets”

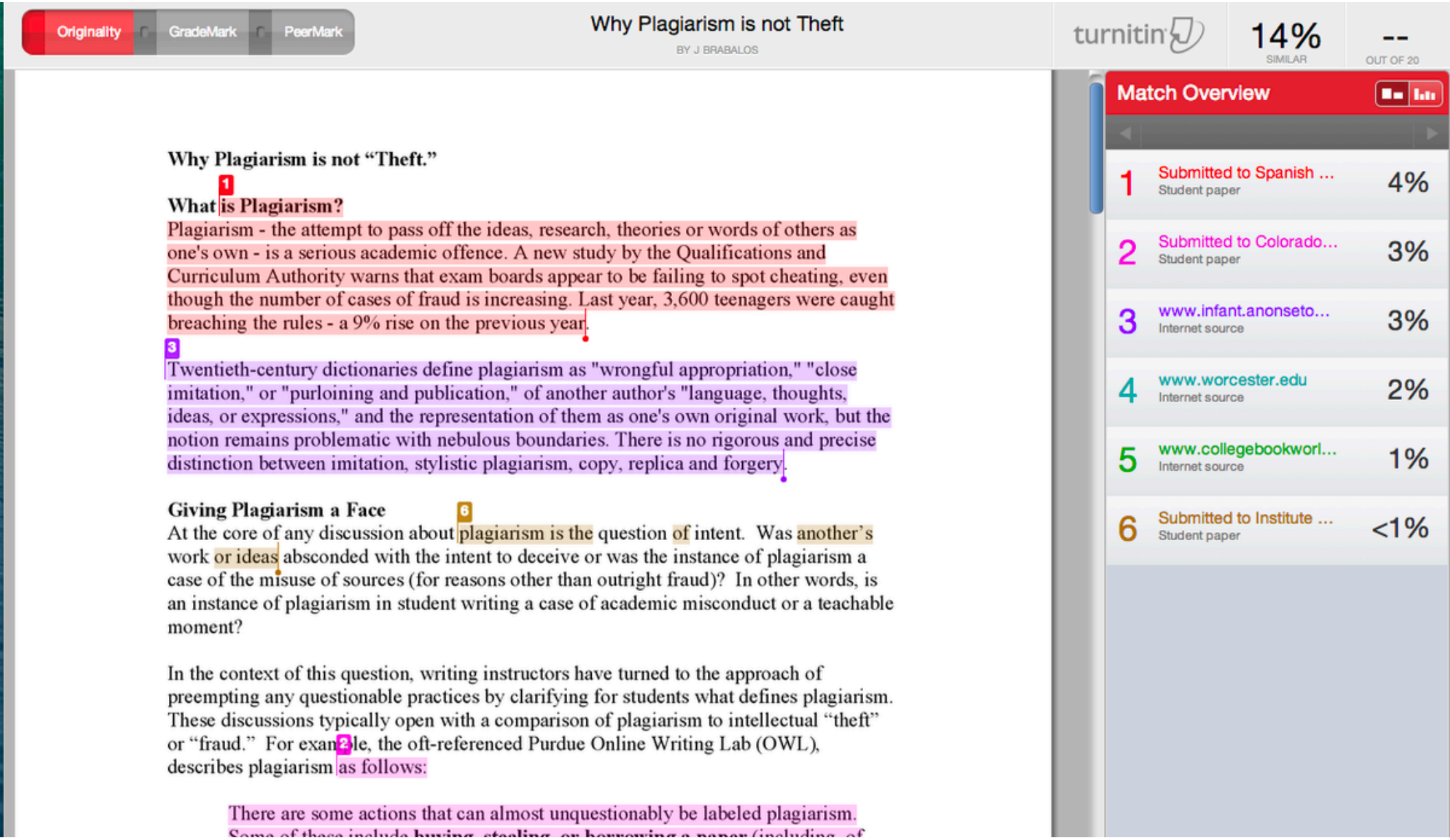
Example: Netflix

- Find viewers who watch similar shows

Example: Amazon

- Find customers who purchase similar items

Finding “nearby” documents: plagiarism detection



Find “legitimate” duplicates

For web search engines:

- Find copies of the same press release
- Find mirrors of the same documents, for efficiency

Finding Similar Items (fast)

Finding similar items fast?

- **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
- **MAGIC: This can be done in** $O(N)$**!! How?**

Distance Measures

- **Goal: Find near-neighbors in high-dim. 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**
  - The **Jaccard similarity** of two **sets** is the size of their intersection divided by the size of their union:
    $sim(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|$
  - **Jaccard distance:** $d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|$

Task: Finding Similar Documents

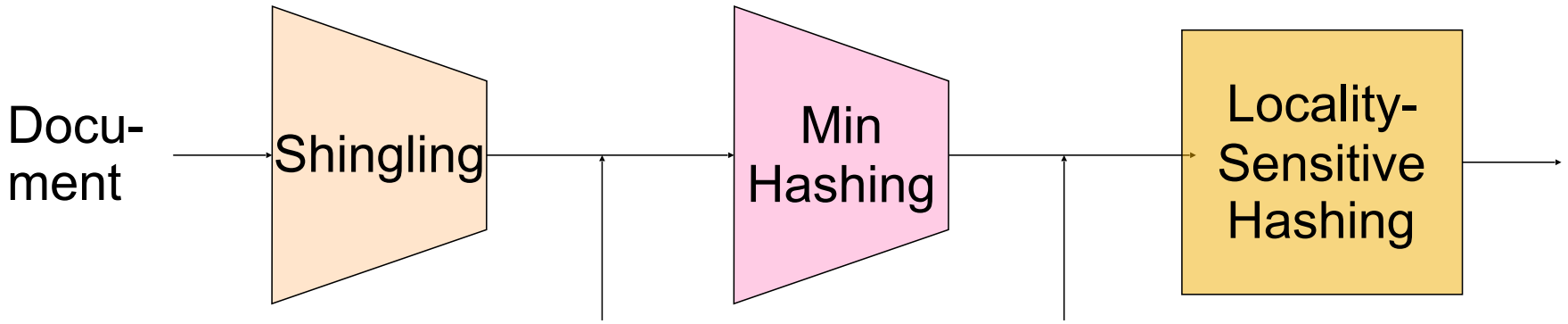
- **Goal: Given a large number (N in the millions or billions) of documents, find "near duplicate" pairs**
- **Applications:**
  - Mirror websites, or approximate mirrors
    - Don't want to show both in search results
  - Similar news articles at many news sites
    - Cluster articles by "same story"
- **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

Main Ideas

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



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

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

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

Shingling

(convert documents to sets)

Naive 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 D1 = abcab; Set of 2-shingles: S(D1) = {ab, bc, ca}

Option: Shingles as a bag (multiset), count ab twice: S'(D1) = {ab, bc, ca, ab}

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

“All hail to dear old Texas A&M, Rally around Maroon and White”

Shingles = {

all hail to dear

hail to dear old

to dear old Texas

dear old texas a&m

old texas a&m rally

…

…

around maroon and white

}

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 shingles: $\mathbf{h(D_1)}$ = {1, 5, 7}

Documents as sets of shingles

A document is now a set of shingles

Assumption: documents with many shingles in common have similar text (even if the text is in a different order)

Caveat: depends on choice of k, otherwise many documents will share lots of shingles

Similarity Metrics 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:**

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

Using shingles directly

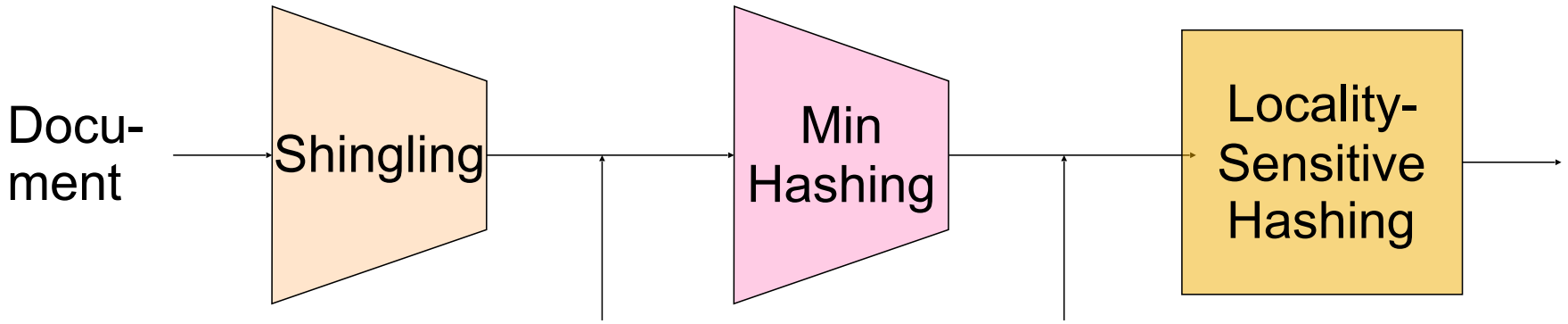
- **Suppose we need to find near-duplicate documents among $N = 1$ million documents**

- Naïvely, we would have to compute **pairwise Jaccard similarities** for **every pair of docs**
  - $N(N-1)/2$ ≈ 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…

Min-hashing

(Convert large sets to short signatures,
while preserving 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

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

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: $\mathbf{d(C_1,C_2)}$ = 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!**

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

Each document is a column

Example: sim(C1 ,C2) = ?

Size of intersection = 3

Size of union = 6

Jaccard similarity (not distance) = 3/6

d(C1,C2) = 1 – (Jaccard similarity) = 3/6

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

Outline: Finding Similar Columns

So far:

- Documents → Sets of shingles
- Represent sets as boolean vectors in a matrix

Next goal: Find similar columns while computing small signatures

- **Similarity of columns == similarity of signatures**

From columns to signatures

We don’t want to compare columns c1, c2, since they might be too large, slowing down the computation

Instead, we compute signatures h(c1), h(c2) that are smaller in size than c1 and c2

The hope:

If c1=c2, then prob(h(c1)=h(c2)) is large

If c1≠c2, then prob(h(c1)=h(c2)) is small

Finding Similar Columns

- **Next Goal: Find similar columns, Small signatures**
- **Naïve approach:**
  - **1) Signatures of columns:** small summaries of columns
  - **2) Examine pairs of signatures** to find similar columns
    - **Essential:** Similarities of signatures and columns are related
  - **3) Optional:** Check 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)

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(·)* 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) ≠ *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

Min-Hashing Example

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

1	5	1	
2	3	1	
6	4	6	

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

Permutation π

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

Input matrix (Shingles x Documents)

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

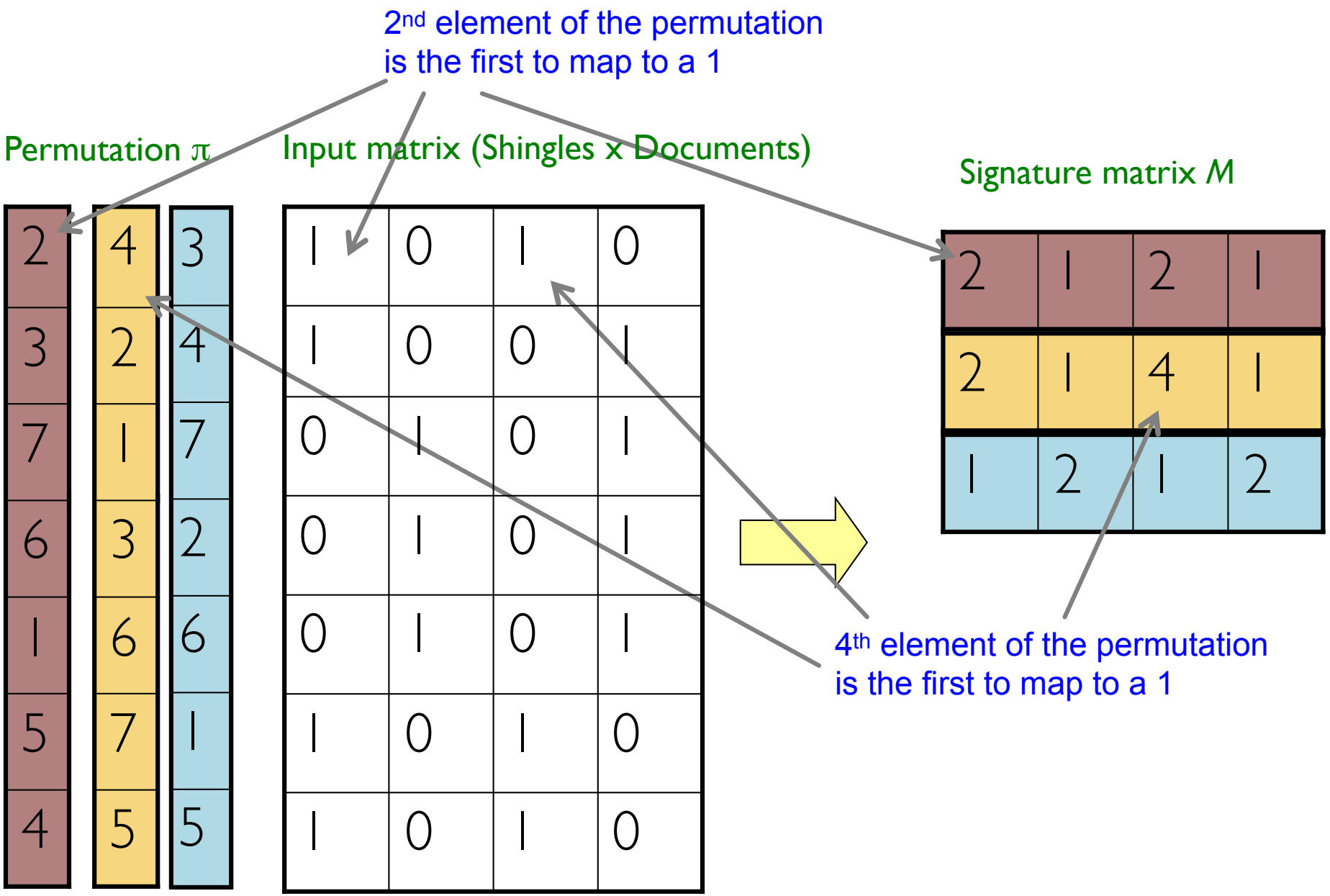
Signature matrix *M*

2	1	2	1

Min-Hashing Example

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

1	5	1	
2	3	1	
6	4	6	



Min-hash approximates Jaccard

- **Choose a random permutation π**
- **Claim:** $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
- **Why?**
  - Let **X** be a doc (set of shingles), $y \in X$ is a shingle
  - **Then:** $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$
    - It is equally likely that any $y \in X$ 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. $y \in C_1 \cap C_2$
  - $\Pr[\min(\pi(C_1))=\min(\pi(C_2))] = |C_1 \cap C_2| / |C_1 \cup C_2| = sim(C_1, C_2)$

Similarity for Signatures

- We know: $\mathbf{Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)}$
- Now generalize to multiple 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

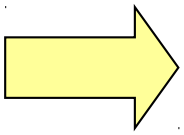
Example: Three Permutations

Permutation π

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

Input matrix (Shingles x Documents)

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

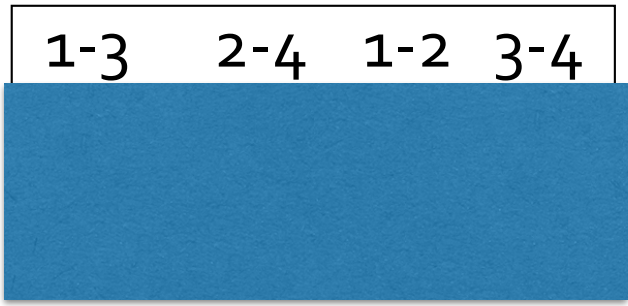


Signature matrix *M*

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

Similarities:

Complete Signatures



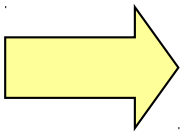
Example: Three Permutations

Permutation π

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

Input matrix (Shingles x Documents)

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



Signature matrix *M*

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

Similarities:

Complete Signatures

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

Minhash 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**