BBS654
Data Mining

Pinar Duygulu

Slides are adapted from

J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, http://www.mmds.org

And

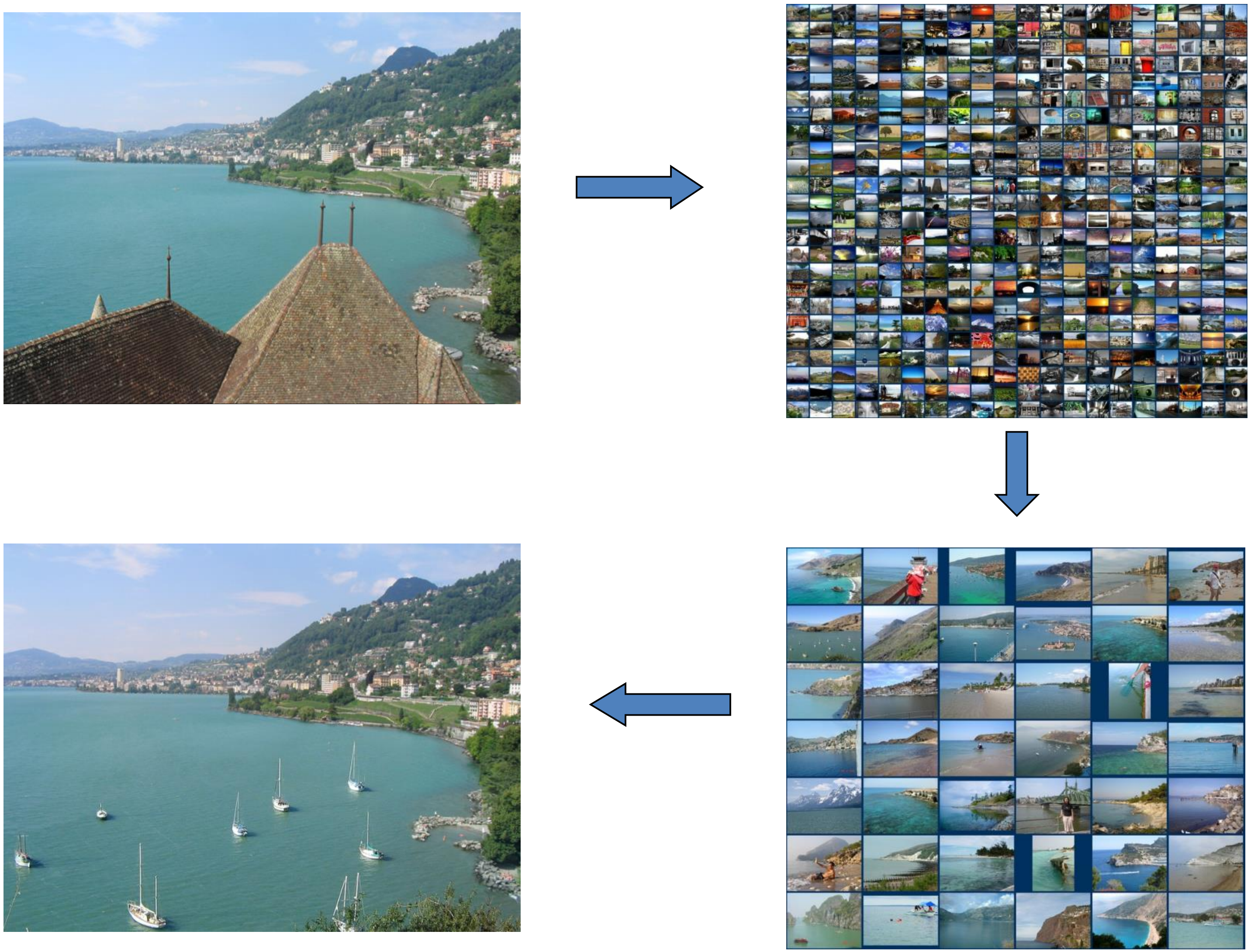
Mustafa Ozdal

CS425: Algorithms for Web Scale Data

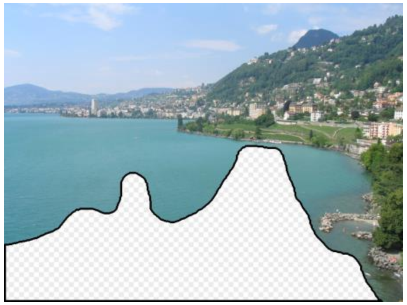
Lecture 3: Similarity Modeling

www.mmds.org

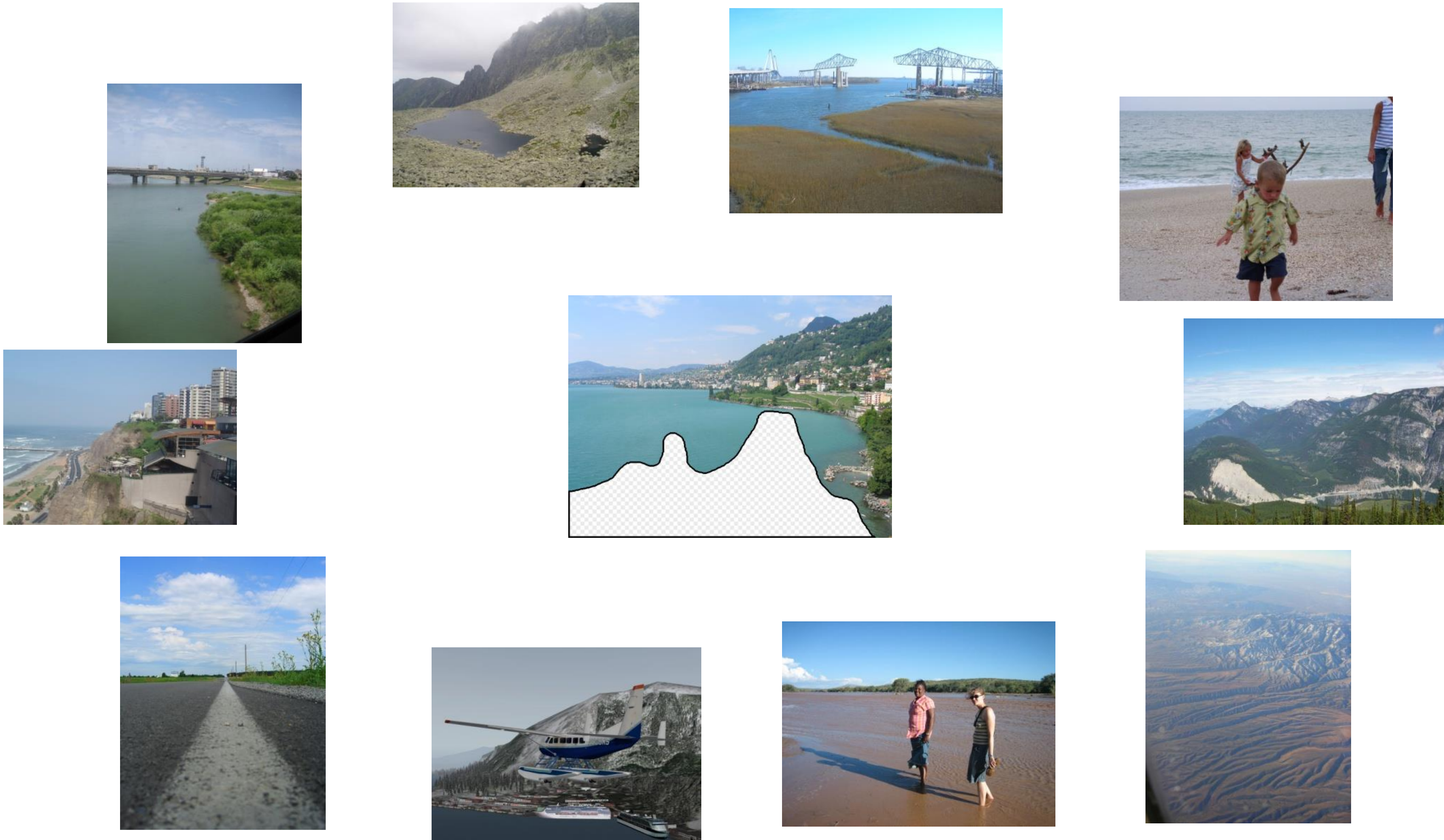
Scene Completion Problem



Scene Completion Problem

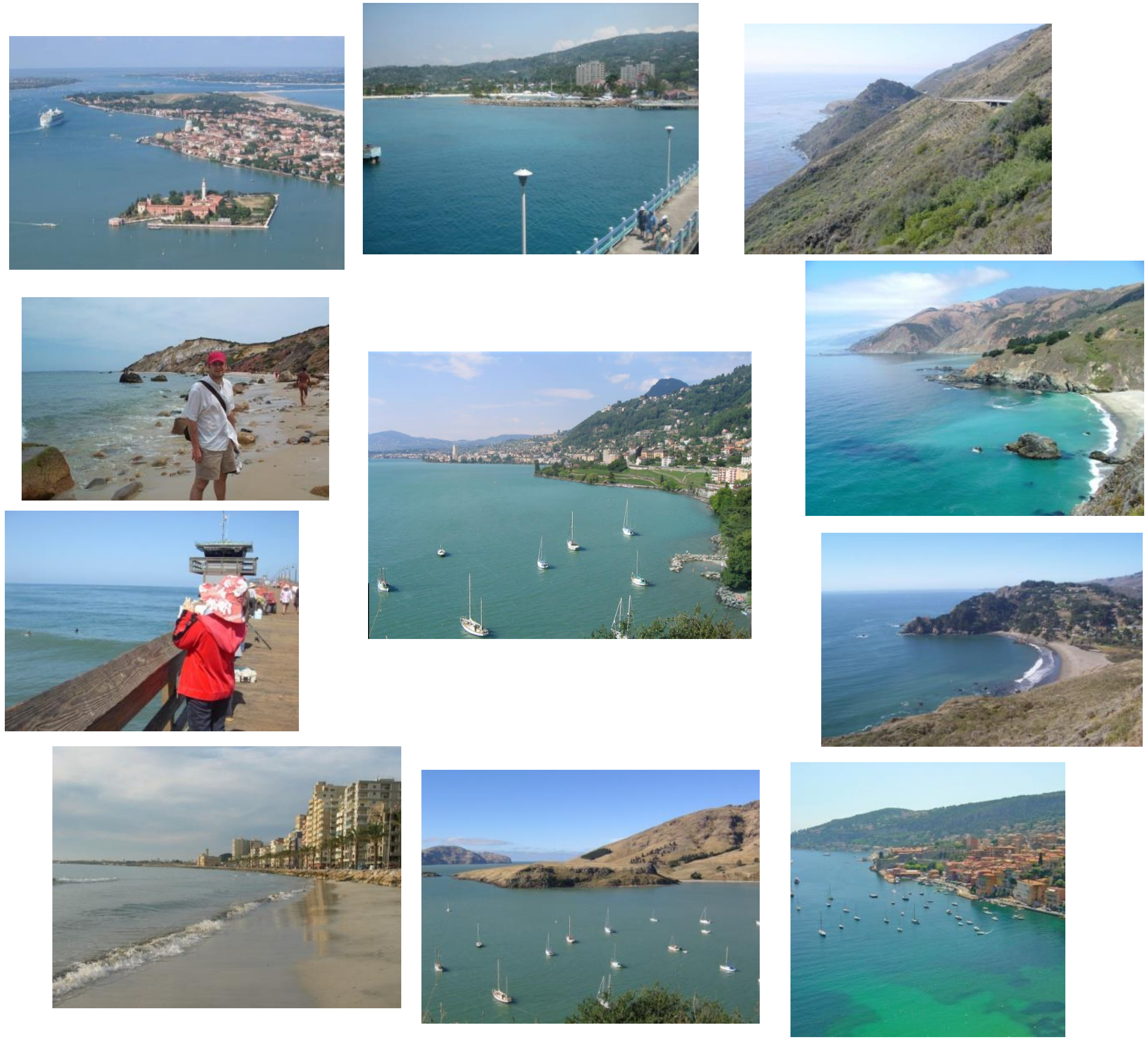


Scene Completion Problem



10 nearest neighbors from a collection of 20,000 images

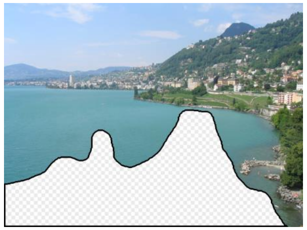
Scene Completion Problem



10 nearest neighbors from a collection of 2 million

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
  - **Customers who purchased similar products**
    - Products with similar customer sets
  - **Images with similar features**



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

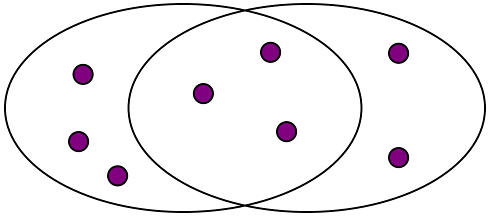
$$\begin{bmatrix} 1 & 2 & 1 \\ 0 & 2 & 1 \\ 0 & 1 & 0 \end{bmatrix} \rightarrow [1\ 2\ 1\ 0\ 2\ 1\ 0\ 1\ 0]$$

- **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 $d(x_i, x_j) \le s$
- **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?**

Finding Similar Items

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: $\boldsymbol{sim(C_1, C_2) = |C_1 \cap C_2| / |C_1 \cup C_2|}$
  - **Jaccard distance:** $\boldsymbol{d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|}$



3 in intersection
8 in union
Jaccard similarity= 3/8
Jaccard distance = 5/8

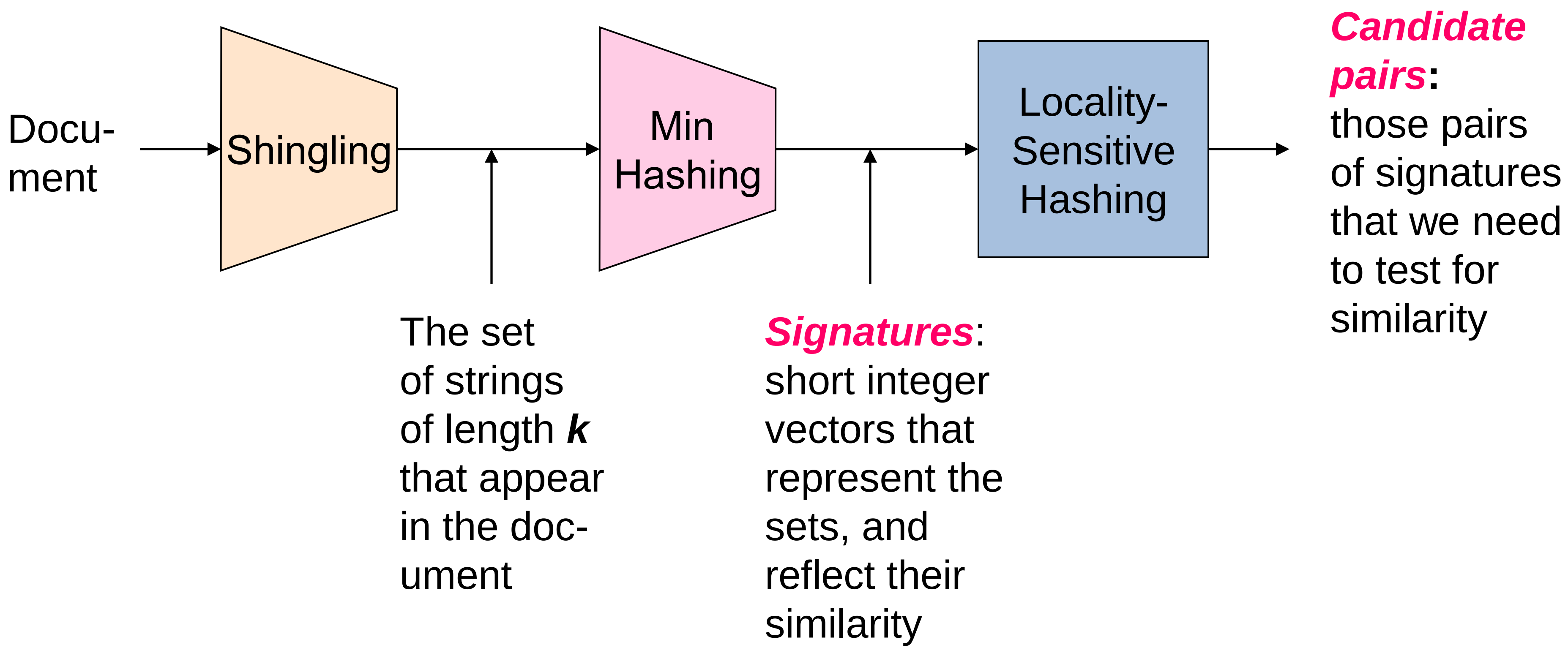
Task: Finding Similar Documents

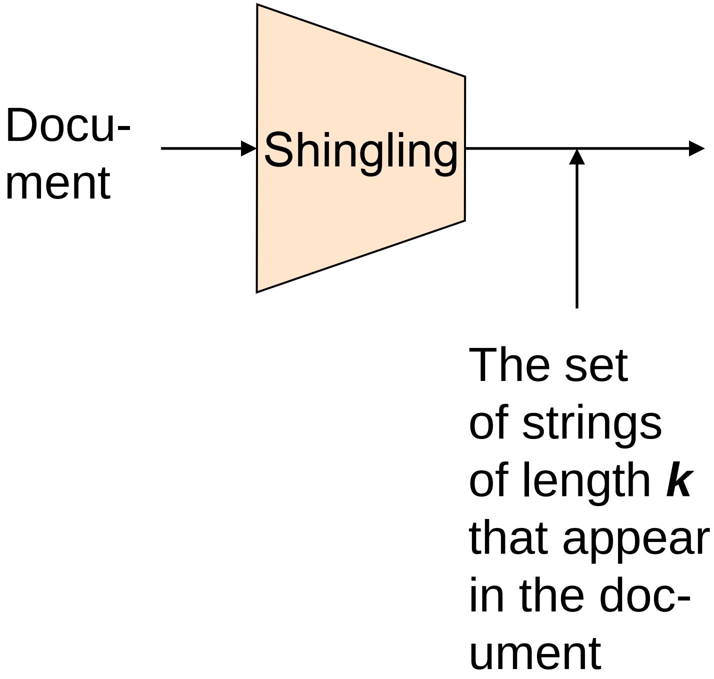
- **Goal: Given a large number (N in the millions or billions) of documents, find "near duplicate" pairs**
- **Applications:**
  - Mirror websites, or approximate mirrors
    - Don't want to show both in search results
  - Similar news articles at many news sites
    - Cluster articles by "same story"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

3 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





Shingling

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

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}

Examples

- Input text:

  “The most effective way to represent documents as sets is to construct from the document the set of short strings that appear within it.”

- 5-shingles:

  “The m”, “he mo”, “e mos”, “ most”, “ ost ”, “ost e”, “st ef”, “t eff”, “ effe”, “effec”, “ffect”, “fecti”, “ectiv”, …

- 9-shingles:

  “The most ”, “he most e”, “e most ef”, “ most eff”, “most effe”, “ost effec”, “st effect”, “t effecti”, “ effectiv”, “effective”, …

Hashing Shingles

- Storage of k-shingles: k bytes per shingle
- Instead, hash each shingle to a 4-byte integer.
  - E.g. "The most " → 4320
    "he most e" → 56456
    "e most ef" → 214509

- Which one is better?
  1. Using 4 shingles?
  2. Using 9-shingles, and then hashing each to 4 byte integer?

- Consider the # of distinct elements represented with 4 bytes

Hashing Shingles

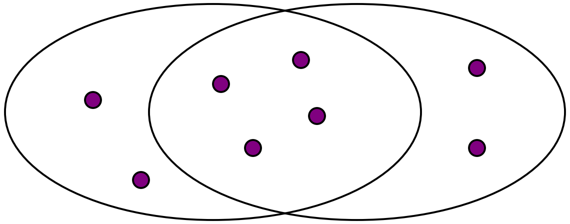
- Not all characters are common.
  - e.g. Unlikely to have shingles like “zy%p”
- Rule of thumb: # of k-shingles is about 20^k

- Using 4-shingles:
  - # of shingles: $20^4 = 160K$
- Using 9-shingles and then hashing to 4-byte values:
  - # of shingles: $20^9 = 512B$
  - # of buckets: $2^{32} = 4.3B$
  - 512B shingles (uniformly) distributed to 4.3B buckets

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:**

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



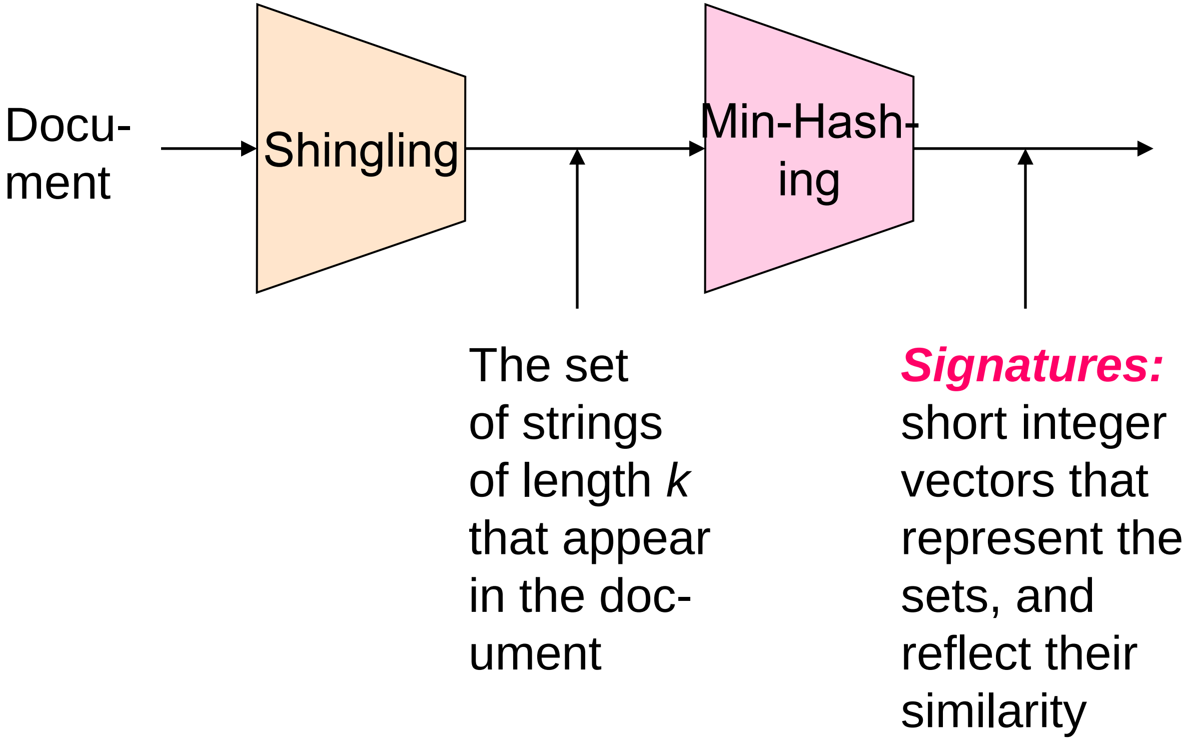
Working Assumption

- **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

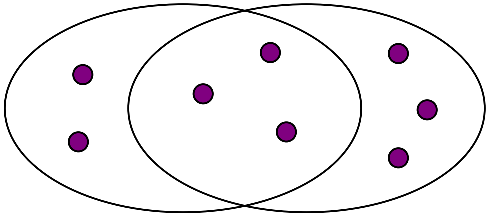
- **Suppose we need to find near-duplicate documents among $N = 1$ million documents**
- Naïvely, we would have to compute **pairwise Jaccard similarities** for **every pair of docs**
  - $N(N-1)/2$ ≈ **5*10¹¹** comparisons
  - At 10^5 secs/day and 10^6 comparisons/sec, it would take **5 days**
- For $N = 10$ million, it takes more than a year…



MinHashing

Step 2: ***Minhashing:*** Convert **large sets** to **short signatures**, while **<u>preserving similarity</u>**

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!**
- **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

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 $\rightarrow$ 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**

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**

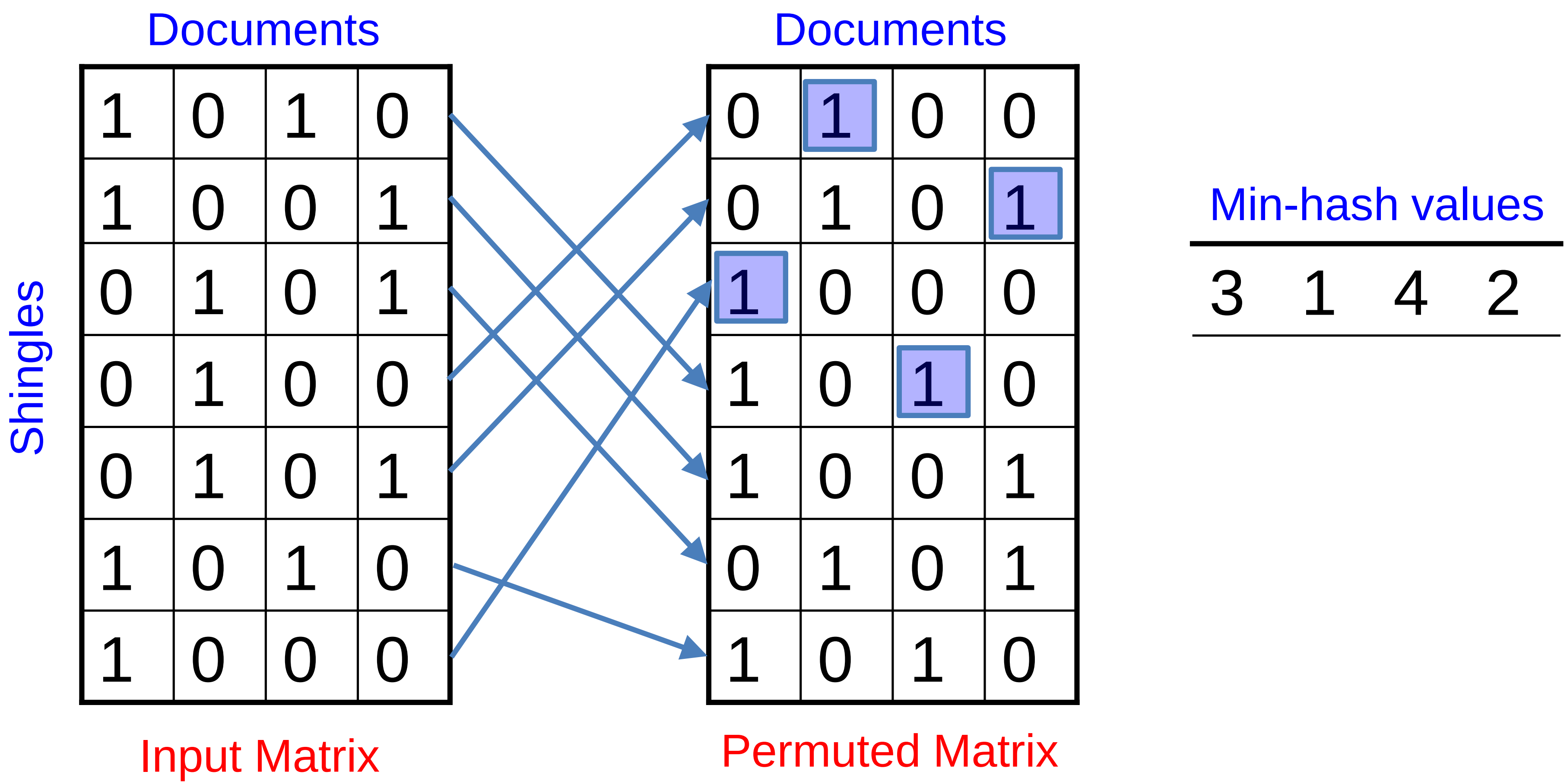
Min-Hashing

- 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

Min-Hashing Example



Min-Hashing Example

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
2	1	4	1
1	2	1	2

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

The Min-Hash Property

- **Choose a random permutation π**
- **Claim:** $\Pr[h_\pi(C_i) = h_\pi(C_j)] = sim(C_i, C_j)$
- **Proof:**
  - Consider 3 types of rows:

    type X: C_i and C_j both have 1s

    type Y: only one of C_i and C_j has 1

    type Z: C_i and C_j both have 0s
  - After random permutation π, what if the first X-type row is before the first Y-type row?

    $h_\pi(C_i) = h_\pi(C_j)$

	C_i		C_j	
X	1		1	
Y	1		0	
Z	0		0	
Z	0		0	
Z	0		0	
X	1		1	
Y	1		0	

Input Matrix

The Min-Hash Property

- What is the probability that the first not-Z row is of type X?

$$\frac{|X|}{|X|+|Y|}$$

□ **Pr[$h_\pi(C_i) = h_\pi(C_j)$] = $\frac{|X|}{|X|+|Y|}$**

- **sim(C_i, C_j) = $\frac{|C_i \cap C_j|}{|C_i \cup C_j|}$ = $\frac{|X|}{|X|+|Y|}$ = Pr[$h_\pi(C_i) = h_\pi(C_j)$]**

- **Conclusion: Pr[$h_\pi(C_i) = h_\pi(C_j)$] = *sim*(C_i, C_j)**

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

Min-Hashing Example

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
2	1	4	1
1	2	1	2

Similarities:

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

Similarity of Signatures

- What is the expected value of Jaccard similarity of two signatures sig_1 and sig_2? Assume there are **s** min-hash values in each signature.

$$E[sim(sig_1, sig_2)] = E\left[\frac{\#\ of\ \pi\ s.t.\ h_\pi(C_1) = h_\pi(C_2)}{s}\right]$$
$$= \frac{1}{s}\sum_{=1}^{s} \Pr[h_\pi(C_1) = h_\pi(C_2)]$$
$$= sim(C_1, C_2)$$

- *Law of large numbers: Average of the results obtained from a large number of trials should be close to the expected value, and will tend to become closer as more trials are performed.*

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 **~400 bytes!**

- **We achieved our goal! We "compressed" long bit vectors into short signatures**

Implementation Trick

- **Permuting rows even once is prohibitive**
- **Row hashing!**
  - Pick **K = 100** hash functions $\boldsymbol{k_i}$
  - Ordering under $\boldsymbol{k_i}$ gives a random row (almost) permutation!

Row	D_1	D_2	D_3	D_4	Hash func. 1 (r+1) % 5	Hash func. 2 (3r+1) % 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

How to pick a random hash function h(x)?
Universal hashing:
$h_{a,b}(x)=((a \cdot x+b) \bmod p) \bmod N$
where:
a,b … random integers
p … prime number (p > N)

Implementation Trick

- **One-pass implementation**
  - For each column ***C*** and hash-func. k_i keep a "slot" for the min-hash value
  - Initialize all $sig(C)[i] = \infty$
  - **Scan rows looking for 1s**
    - Suppose row ***j*** has 1 in column ***C***
    - Then for each k_i :
      - If $k_i(j) < sig(C)[i]$, then $sig(C)[i] \leftarrow k_i(j)$

Example: Computing Min-Hash Signatures

Row	D_1	D_2	D_3	D_4	Hash func. 1 (r+1) % 5	Hash func. 2 (3r+1) % 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

Signatures

D_1	D_2	D_3	D_4
∞	∞	∞	∞
∞	∞	∞	∞

Example: Computing Min-Hash Signatures

Row	D_1	D_2	D_3	D_4	Hash func. 1 (r+1) % 5	Hash func. 2 (3r+1) % 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

Signatures

D_1	D_2	D_3	D_4
1	∞	∞	1
1	∞	∞	1

Example: Computing Min-Hash Signatures

Row	D_1	D_2	D_3	D_4	Hash func. 1 (r+1) % 5	Hash func. 2 (3r+1) % 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

Signatures

D_1	D_2	D_3	D_4
1	∞	2	1
1	∞	4	1

Example: Computing Min-Hash Signatures

Row	D_1	D_2	D_3	D_4	Hash func. 1 (r+1) % 5	Hash func. 2 (3r+1) % 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

Signatures

D_1	D_2	D_3	D_4
1	3	2	1
1	2	4	1

Example: Computing Min-Hash Signatures

Row	D_1	D_2	D_3	D_4	Hash func. 1 (r+1) % 5	Hash func. 2 (3r+1) % 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

Signatures

D_1	D_2	D_3	D_4
1	3	2	1
0	2	0	0

Example: Computing Min-Hash Signatures

Row	D_1	D_2	D_3	D_4	Hash func. 1 (r+1) % 5	Hash func. 2 (3r+1) % 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

Signatures

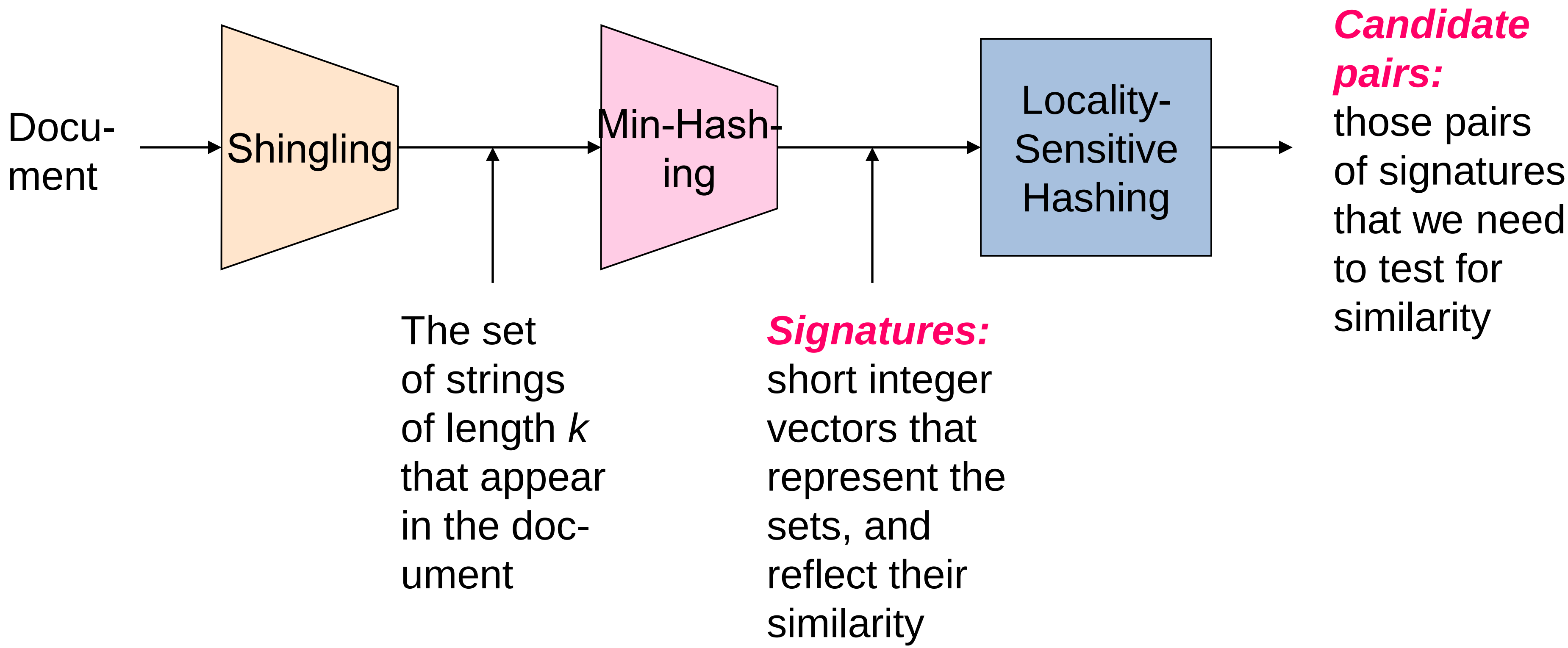
D_1	D_2	D_3	D_4
1	3	0	1
0	2	0	0

Example: Computing Min-Hash Signatures

Row	D_1	D_2	D_3	D_4	Hash func. 1 (r+1) % 5	Hash func. 2 (3r+1) % 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

Final signatures

D_1	D_2	D_3	D_4
1	3	0	1
0	2	0	0



Locality Sensitive Hashing

Step 3: ***Locality-Sensitive Hashing:***
Focus on pairs of signatures likely to be from similar documents

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**

LSH for Min-Hash

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

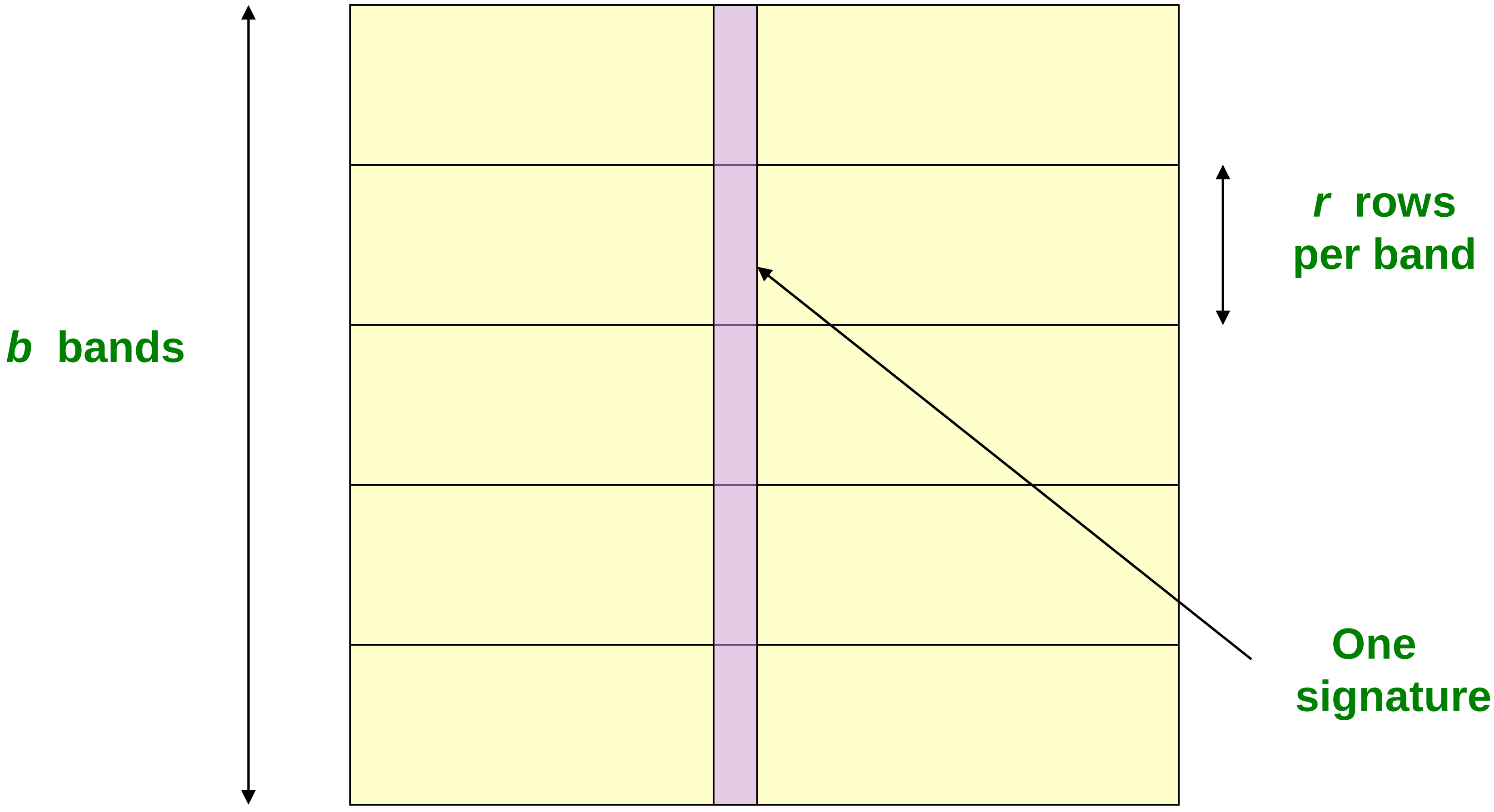
- **Big idea: Hash columns of signature matrix *M* several times**

- 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**

Partition *M* into *b* Bands

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

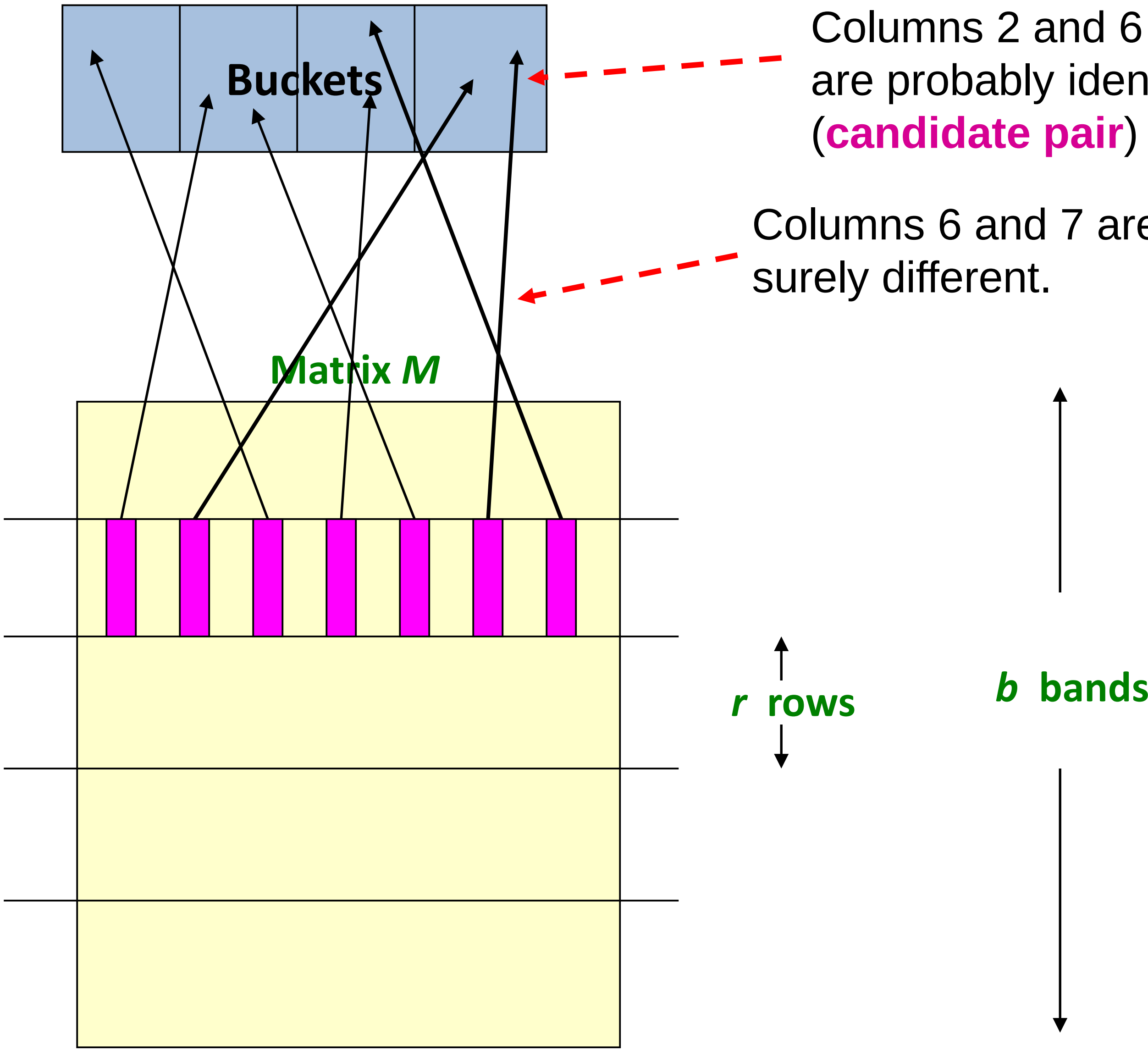


Signature matrix *M*

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

Hashing Bands



Columns 2 and 6 are probably identical (**candidate pair**)

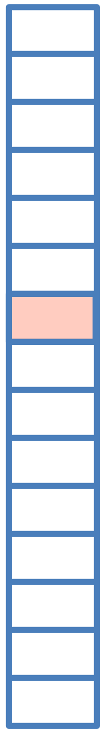
Columns 6 and 7 are surely different.

Banding Example

Signature Matrix

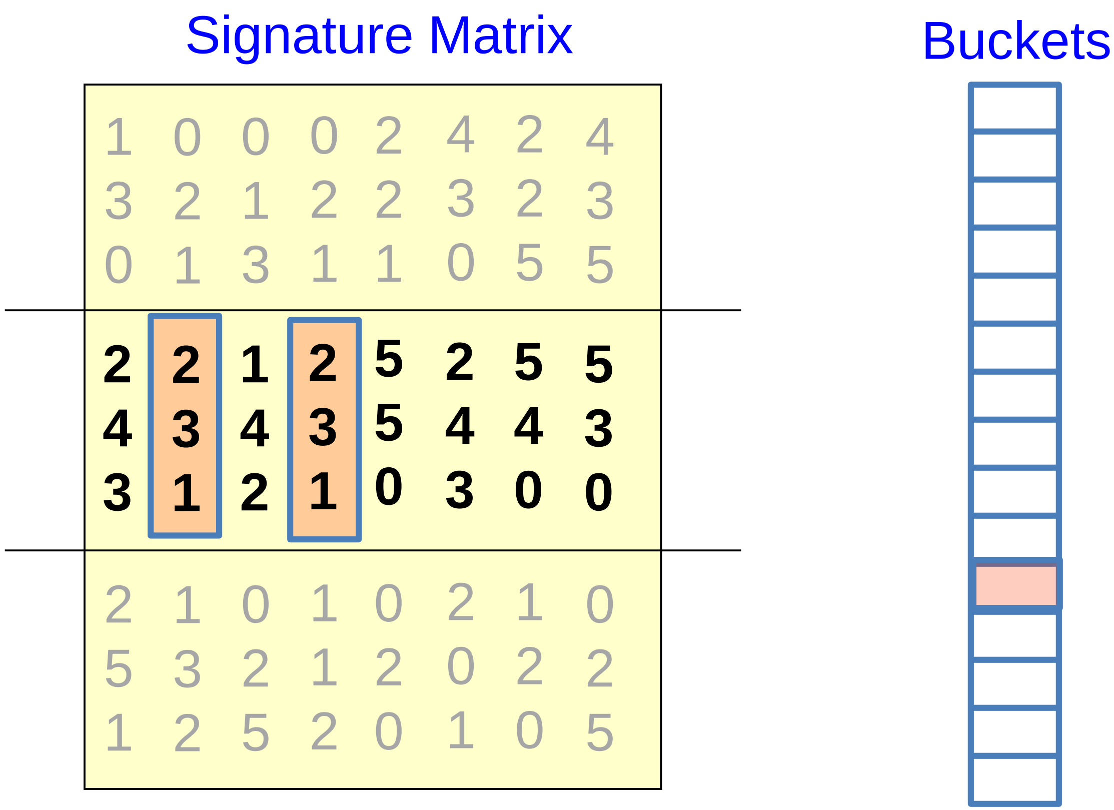
1	0	0	0	2	4	2	4
3	2	1	2	2	3	2	3
0	1	3	1	1	0	5	5
2	2	1	2	5	2	5	5
4	3	4	3	5	4	4	3
3	1	2	1	0	3	0	0
2	1	0	1	0	2	1	0
5	3	2	1	2	0	2	2
1	2	5	2	0	1	0	5

Buckets



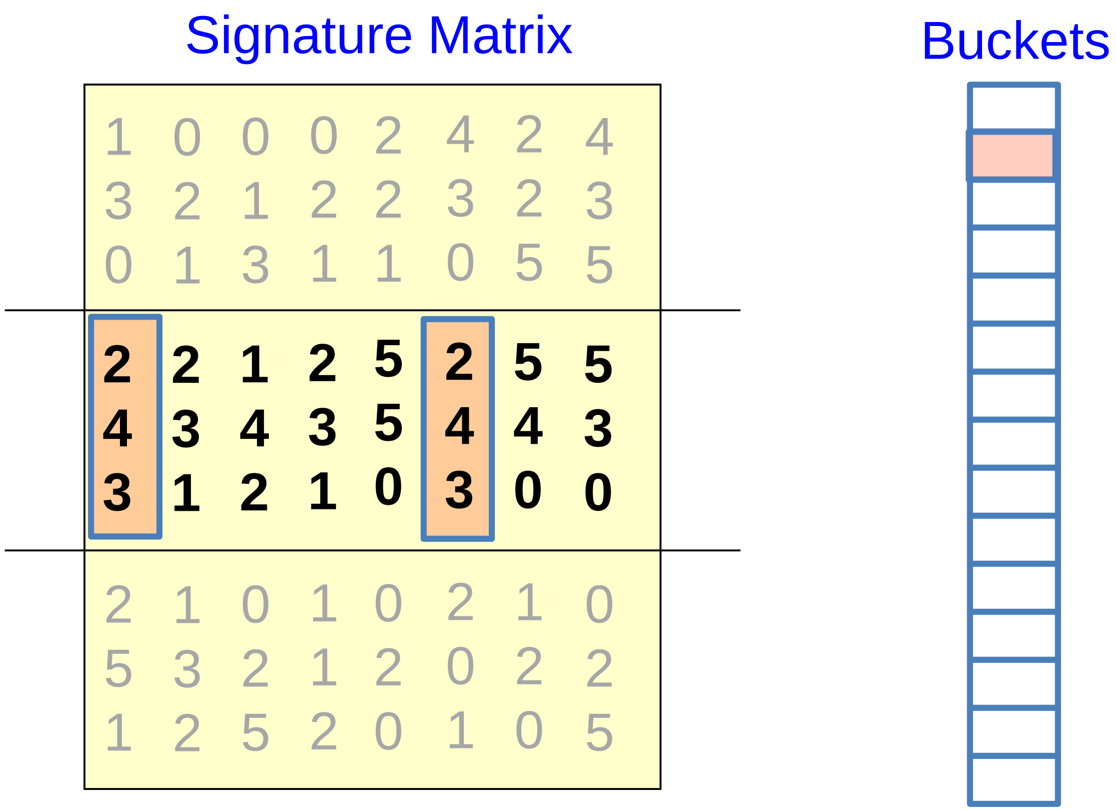
Candidate pairs: {(2,4);

Banding Example



Candidate pairs: {(2,4);

Banding Example



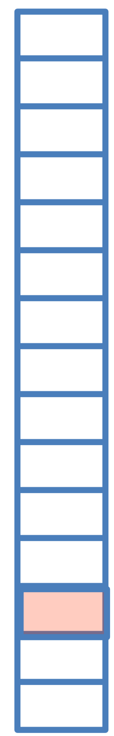
Candidate pairs: {(2,4); (1,6)

Banding Example

Signature Matrix

1	0	0	0	2	4	2	4
3	2	1	2	2	3	2	3
0	1	3	1	1	0	5	5
2	2	1	2	5	2	5	5
4	3	4	3	5	4	4	3
3	1	2	1	0	3	0	0
2	**1**	**0**	**1**	**0**	**2**	**1**	**0**
5	**3**	**2**	**1**	**2**	**0**	**2**	**2**
1	**2**	**5**	**2**	**0**	**1**	**0**	**5**

Buckets



Candidate pairs: {(2,4); (1,6) (3,8)}

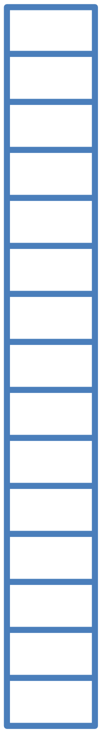
Banding Example

Signature Matrix

1	0	0	0	2	4	2	4
3	2	1	2	2	3	2	3
0	1	3	1	1	0	5	5
2	2	1	2	5	2	5	5
4	3	4	3	5	4	4	3
3	1	2	1	0	3	0	0
2	1	0	1	0	2	1	0
5	3	2	1	2	0	2	2
1	2	5	2	0	1	0	5

True positive

Buckets



Candidate pairs: {(2,4); (1,6); (3,8)}

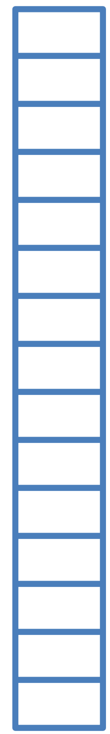
Banding Example

Signature Matrix

1	0	0	0	2	4	2	4
3	2	1	2	2	3	2	3
0	1	3	1	1	0	5	5
2	2	1	2	5	2	5	5
4	3	4	3	5	4	4	3
3	1	2	1	0	3	0	0
2	1	0	1	0	2	1	0
5	3	2	1	2	0	2	2
1	2	5	2	0	1	0	5

True positive

Buckets



Candidate pairs: {(2,4); (1,6); (3,8)}

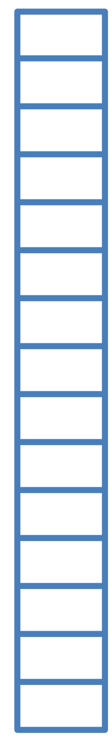
Banding Example

Signature Matrix

1	0	0	0	2	4	2	4
3	2	1	2	2	3	2	3
0	1	3	1	1	0	5	5
2	2	1	2	5	2	5	5
4	3	4	3	5	4	4	3
3	1	2	1	0	3	0	0
2	1	0	1	0	2	1	0
5	3	2	1	2	0	2	2
1	2	5	2	0	1	0	5

False positive?

Buckets



Candidate pairs: {(2,4); (1,6); (3,8)}

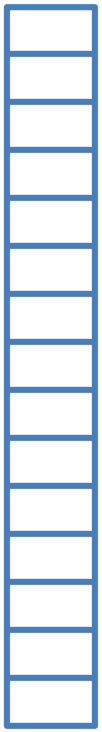
Banding Example

Signature Matrix

1	0	0	0	2	4	2	4
3	2	1	2	2	3	2	3
0	1	3	1	1	0	5	5
2	2	1	2	5	2	5	5
4	3	4	3	5	4	4	3
3	1	2	1	0	3	0	0
2	1	0	1	0	2	1	0
5	3	2	1	2	0	2	2
1	2	5	2	0	1	0	5

False negative?

Buckets



Candidate pairs: {(2,4); (1,6); (3,8)}

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

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

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

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

C_1, C_2 are 80% Similar

- **Find pairs of** $\geq$ ***s***=0.8 similarity, set **b**=20, **r**=5
- **Assume:** sim(C_1, C_2) = 0.8
  - 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 ***different*** 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 miss them)
  - **We would find 99.965% pairs of truly similar documents**

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

C_1, C_2 are 30% Similar

- **Find pairs of** $\geq$ ***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 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

Summary: 3 Steps

- **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 $\mathbf{Pr}[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$
  - 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**

Distance Metrics

Distance Measure

- A distance measure $d(x,y)$ must have the following properties:

1. $d(x,y) \geq 0$
2. $d(x,y) = 0$ iff $x = y$
3. $d(x,y) = d(y,x)$
4. $d(x,y) \leq d(x,z) + d(z,y)$

Euclidean Distance

- Consider two items x and y with n numeric attributes

- Euclidean distance in n-dimensions:

$$d([x_1, x_2, \ldots, x_n], [y_1, y_2, \ldots, y_n]) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

- Useful when you want to penalize larger differences more than smaller ones

L_r- Norm

- Definition of L_r-norm:

$$d([x_1, x_2, \dots, x_n], (y_1, y_2, \dots, y_n)] = \left(\sum_{i=1}^{n} |x_i - y_i|^r\right)^{1/r}$$

- Special cases:
  - **L_1-norm:** Manhattan distance
    - Useful when you want to penalize differences in a linear way (e.g. a difference of 10 for one attribute is equivalent to difference of 1 for 10 attributes)
  - **L_2-norm:** Euclidean distance
  - **L_∞-norm:** Maximum distance among all attributes
    - Useful when you want to penalize the largest difference in an attribute

Jaccard Distance

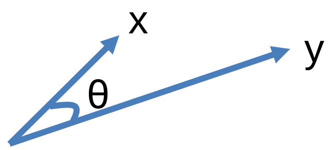
- Given two sets x and y:

$$d(x, y) = 1 - \frac{|x \cap y|}{|x \cup y|}$$

- Useful for set representations
  - i.e. An element either exists or does not exist
- What if the attributes are weighted?
  - e.g. Term frequency in a document

Cosine Distance

- Consider x and y represented as vectors in an n-dimensional space

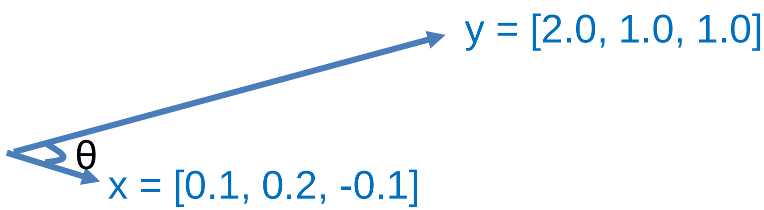


$$\cos(\theta) = \frac{x.y}{||x||.||y||}$$

- The cosine distance is defined as the θ value
  - Or, cosine similarity is defined as cos(θ)

- Only direction of vectors considered, not the magnitudes
- Useful when we are dealing with vector spaces

Cosine Distance: Example



$$\cos(\theta) = \frac{x.y}{||x||.||y||}$$

$$= \frac{0.2 + 0.2 \; - 0.1}{\sqrt{0.01 + 0.04 + 0.01}\,.\sqrt{4 + 1 + 1}}$$

$$= \frac{0.3}{\sqrt{0.36}} = 0.5 \Rightarrow \theta = 60^0$$

Note: The distance is independent of vector magnitudes

Edit Distance

- What happens if you search for “Blkent” in Google?
  - “Showing results for Bilkent.”
- Edit distance between x and y: Smallest number of insertions, deletions, or mutations needed to go from x to y.
- What is the edit distance between “BILKENT” and “BLANKET”?

B	**I**	**L**		K	E	**N**	T	
B	**L**	**A**	**N**	K	E		T	

B	**I**	L			K	E	**N**	T
B		L	**A**	**N**	K	E		T

dist(BILKENT, BLANKET) = 4

- *Efficient dynamic-programming algorithms exist to compute edit distance (CS473)*

Distance Metrics Summary

- Important to choose the right distance metric for your application
  - Set representation?
  - Vector space?
  - Strings?

- Distance metric chosen also affects complexity of algorithms
  - Sometimes more efficient to optimize L_1 norm than L_2 norm.
  - Computing edit distance for long sequences may be expensive

- Many other distance metrics exist.

Applications of LSH

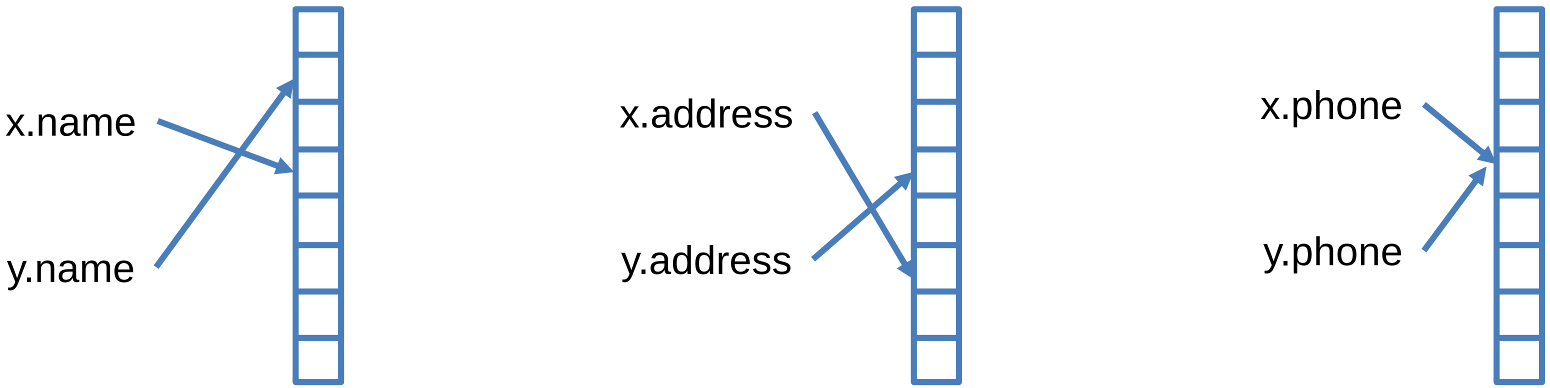
Entity Resolution

Entity Resolution

- Many records exist for the same person with slight variations
  - Name: “Robert W. Carson” vs. “Bob Carson Jr.”
  - Date of birth: “Jan 15, 1957” vs. “1957” vs none
  - Address: Old vs. new, incomplete, typo, etc.
  - Phone number: Cell vs. home vs. work, with or without country code, area code
- Objective: Match the same people in different databases

Locality Sensitive Hashing (LSH)

- Simple implementation of LSH:
  - Hash each field separately
  - If two people hash to the same bucket for any field, add them as a candidate pair



Candidate Pair Evaluation

- Define a scoring metric and evaluate candidate pairs
- Example:
  - Assign a score of 100 for each field. Perfect match gets 100, no match gets 0.
  - Which distance metric for names?
    - Edit distance, but with quadratic penalty
  - How to evaluate phone numbers?
    - Only exact matches allowed, but need to take care of missing area codes.
  - Pick a score threshold empirically and accept the ones above that
    - Depends on the application and importance of false positives vs. negatives
    - Typically need cross validation

Fingerprint Matching

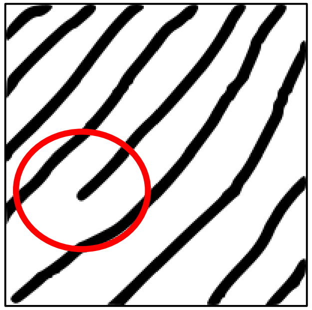
Fingerprint Matching

- Many-to-many matching: Find out all pairs with the same fingerprints
  - Example: You want to find out if the same person appeared in multiple crime scenes
- One-to-many matching: Find out whose fingerprint is on the gun
  - Too expensive to compare even one fingerprint with the whole database
  - Need to use LSH even for one-to-many problem
- Preprocessing:
  - Different sizes, different orientations, different lighting, etc.
  - Need some normalization in preprocessing (not our focus here)

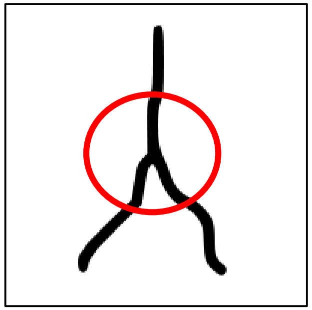
Fingerprint Features

- Minutia: Major features of a fingerprint

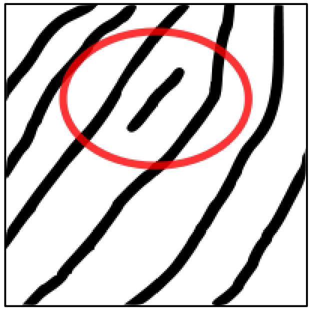
Ridge ending



Bifurcation



Short ridge



...

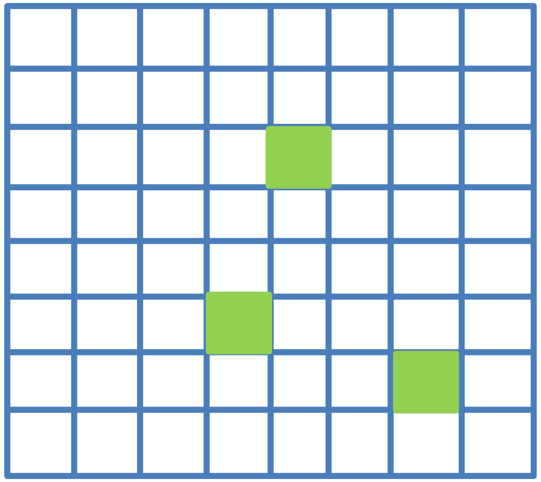
Image Source: Wikimedia Commons

Fingerprint Grid Representation

- Overlay a grid and identify points with minutia

	X					X	
			X				
			X		X		
							X
	X	X					

Special Hash Function



- Choose 3 grid points
- If a fingerprint has minutia in all 3 points, add it to the bucket
- Otherwise, ignore the fingerprint.

Locality Sensitive Hashing

- Define 1024 hash functions
  - i.e. Each hash function is defined as 3 grid points
- Add fingerprints to the buckets hash functions
- If multiple fingerprints are in the same bucket, add them as a candidate pair.

Example

- Assume:
  - Probability of finding a minutia at a random grid point = 20%
  - If two fingerprints belong to the same finger:
    - Probability of finding a minutia at the same grid point = 80%
- For two different fingerprints:
  - Probability that they have minutia at point (x, y)?

    $0.2 * 0.2 = 0.04$
  - Probability that they hash to the same bucket for a given hash function?

    $0.04^3 = 0.000064$
- For two fingerprints from the same finger:
  - Probability that they have minutia at point (x, y)?

    $0.2 * 0.8 = 0.16$
  - Probability that they hash to the same bucket for a given hash function?

    $0.16^3 = 0.004096$

Example (cont'd)

- For two different fingerprints and 1024 hash functions:
  - Probability that they hash to the same bucket at least once?

    $1 - (1\text{-}0.04^3)^{1024} = 0.063$
- For two fingerprints from the same finger and 1024 hash functions:
  - Probability that they hash to the same bucket at least once?

    $1 - (1\text{-}0.16^3)^{1024} = 0.985$

- False positive rate?

  6.3%
- False negative rate?

  1.5%

Example (cont'd)

- How to reduce the false positive rate?
- Try: Increase the number grid points from 3 to 6

- For two different fingerprints and 1024 hash functions:
  - Probability that they hash to the same bucket at least once?

$$1 - (1-0.04^6)^{1024} = 0.0000042$$

- For two fingerprints from the same finger and 1024 hash functions:
  - Probability that they hash to the same bucket at least once?

$$1 - (1-0.16^6)^{1024} = 0.017$$

- False negative rate increased to 98.3%!

Example (cont'd)

- Second try: Add another AND function to the original setting

1. Define 2048 hash functions

   Each hash function is based on 3 grid points as before

2. Define two groups each with 1024 hash functions
3. For each group, apply LSH as before

   Find fingerprints that share a bucket for at least one hash function

4. If two fingerprints share at least one bucket in both groups, add them as a candidate pair

Example (cont'd)

- *Reminder:*
  - *Probability that two fingerprints hash to the same bucket at least once for 1024 hash functions:*
    - *If two different fingerprints:* $1 - (1-0.04^3)^{1024} = 0.063$
    - *If from the same finger:* $1 - (1-0.16^3)^{1024} = 0.985$
- With the AND function at the end:
  - Probability that two fingerprints are chosen as candidate pair:
    - If two different fingerprints:

      $$0.063 \times 0.063 = 0.004$$
    - If from the same finger:

      $$0.985 \times 0.985 = 0.97$$
- Reduced false positives to 0.4%, but increased false negatives to 3%
- What if we add another OR function at the end?