Finding similar items: Locality-sensitive Hashing

COMPCSI 753: Algorithms for Massive Data

Ninh Pham

University of Auckland

Parts of this material are modifications of the lecture slides from

http://mmds.org

Designed for the textbook **Mining of Massive Datasets**

by Jure Leskovec, Anand Rajaraman, and Jeff Ullman.

Auckland, Aug 4, 2020

Recap: Universal hashing

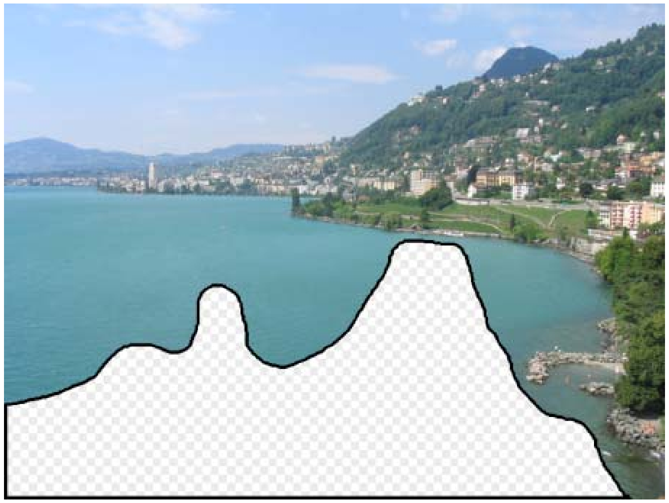
- Given a random choice of a hash function **h** from the universal family such that **h:** $\mathbf{U} \rightarrow \{\mathbf{0, 1, \ldots, n\text{-}1}\}$.
- Our dictionary needs **O(n)** space and **O(1)** expected time per operation (search, insert, delete).
- Expectation is over the random choice of hash function.
- Independent of input set.

Outline

- Similarity search applications
- Finding similar documents:
  - Shingling
  - MinHash
  - Locality-sensitive hashing (LSH)

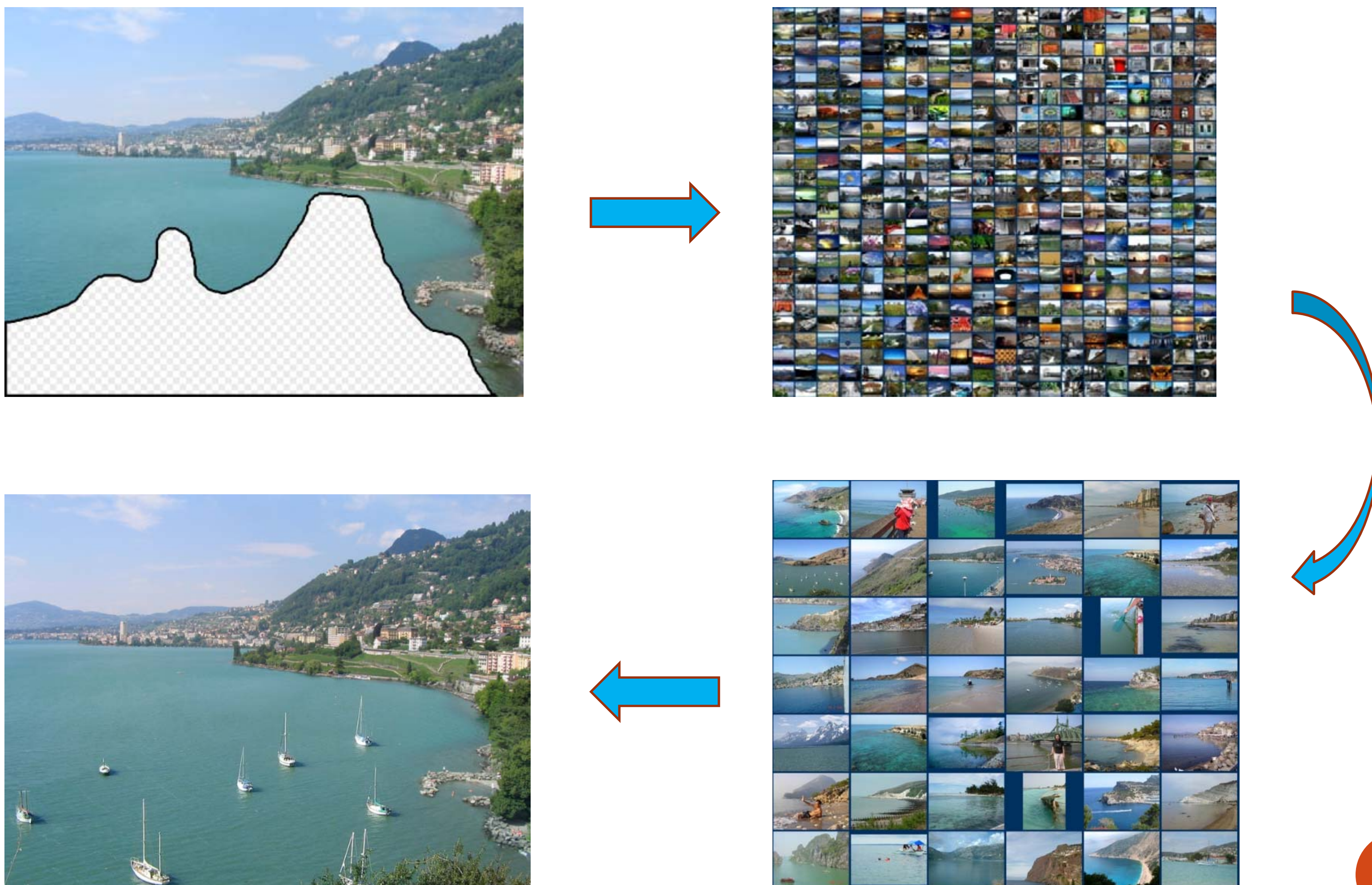
Scene completion problem

- Problem:
  - Patch up holes in an image by finding similar image region from a huge image database on the Web.

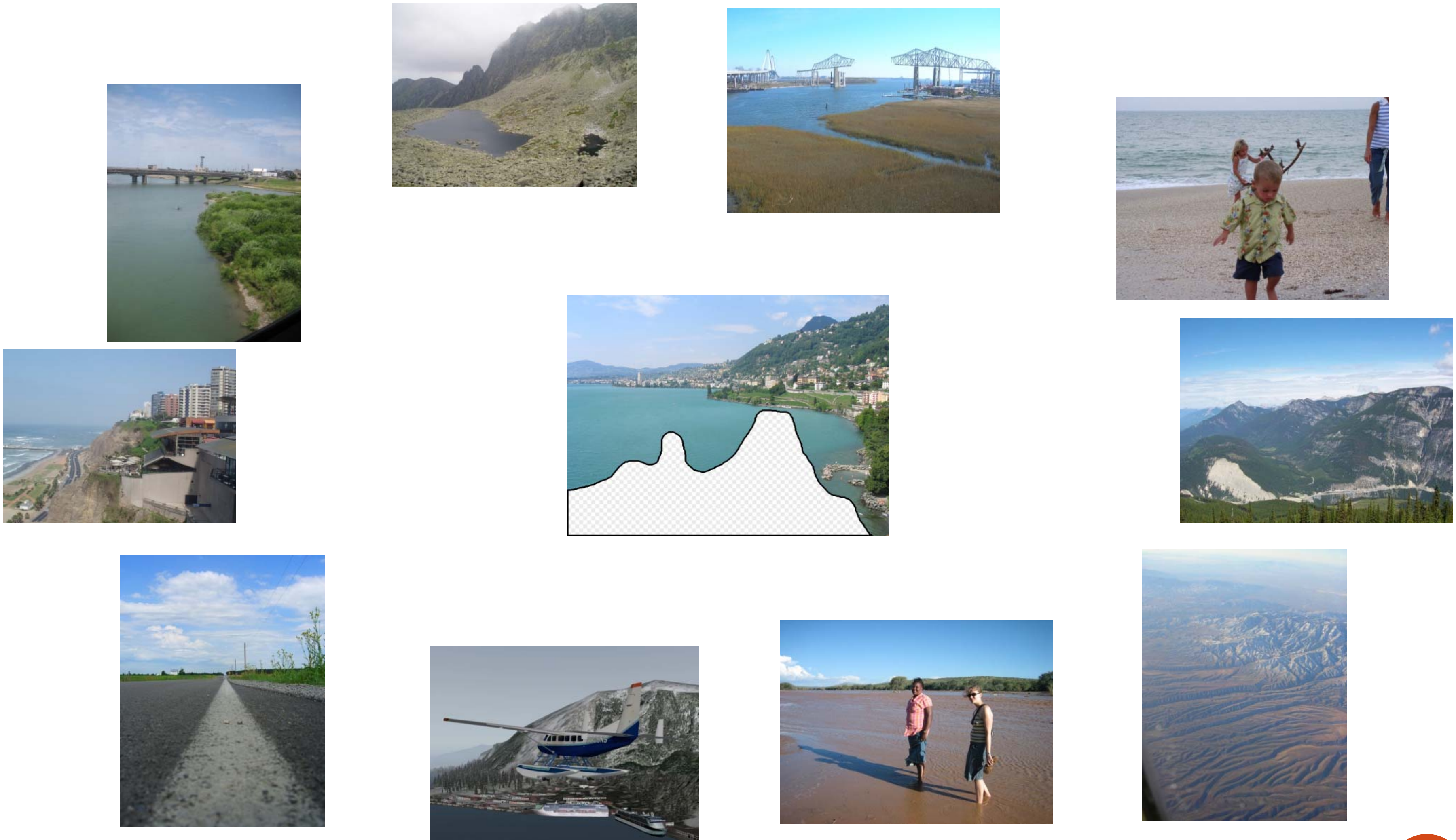


Source: http://graphics.cs.cmu.edu/projects/scene-completion/

Scene completion problem

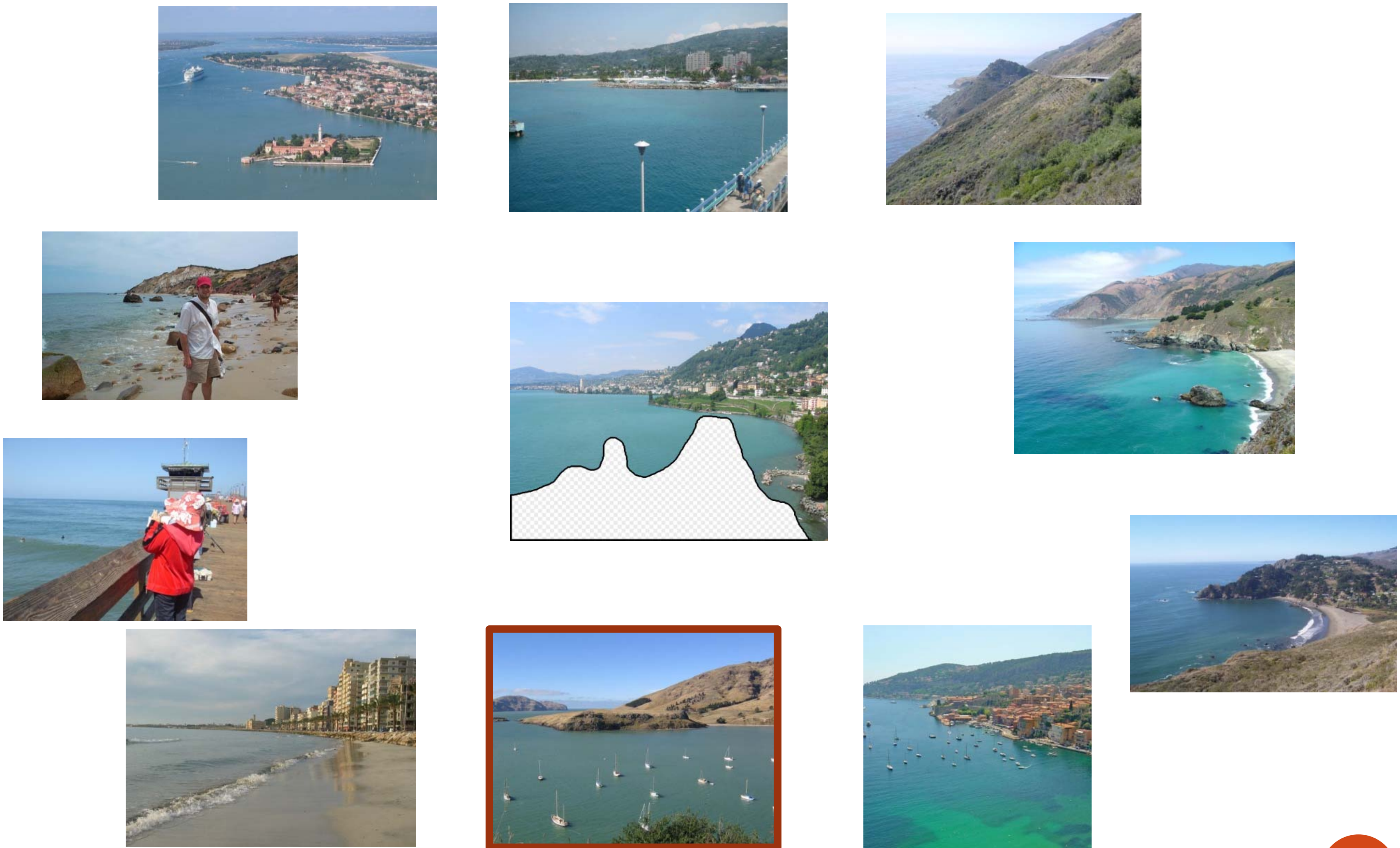


Scene completion problem



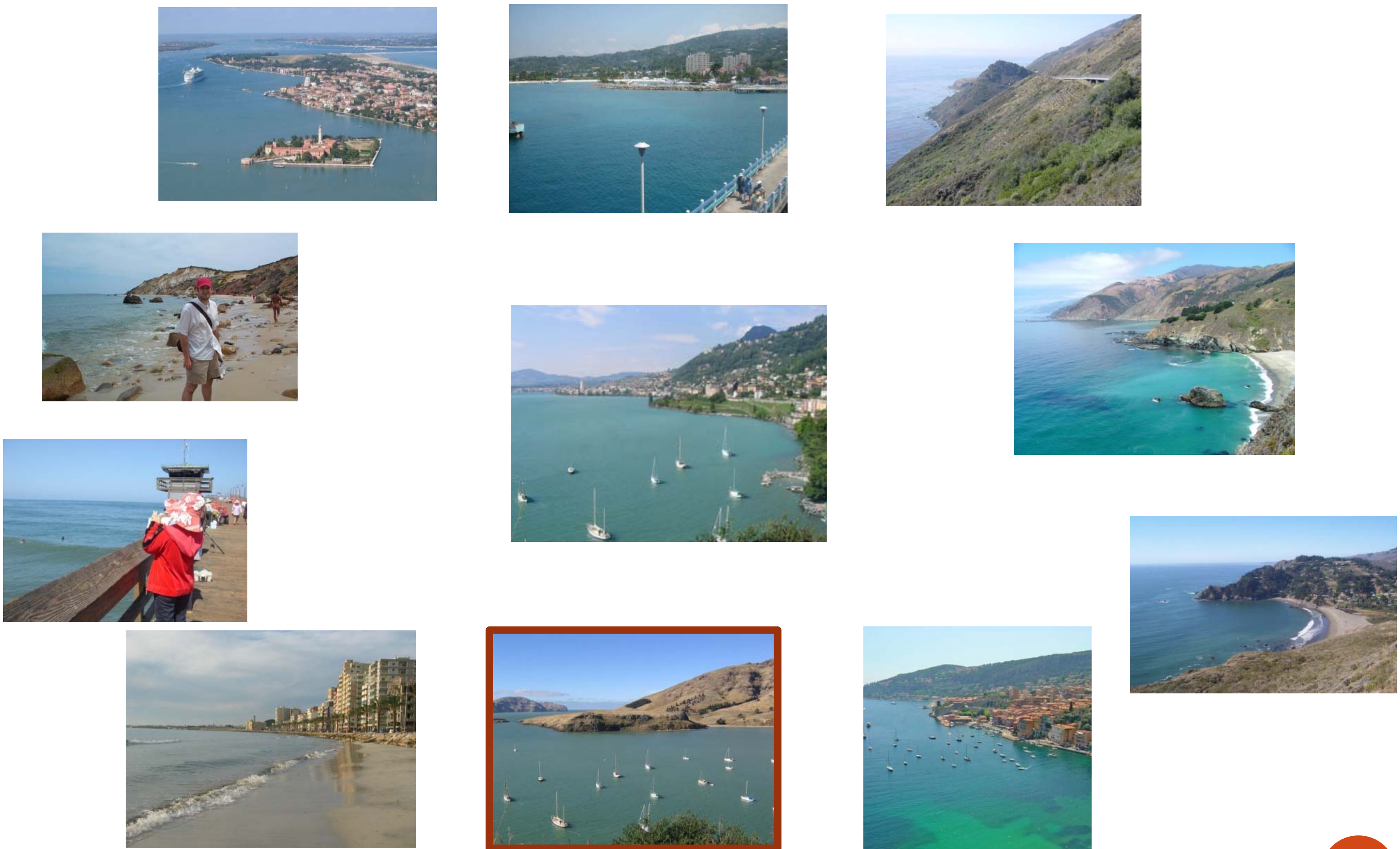
10 nearest neighbors from a collection of **20,000** images

Scene completion problem



10 nearest neighbors from a collection of **2 million** images

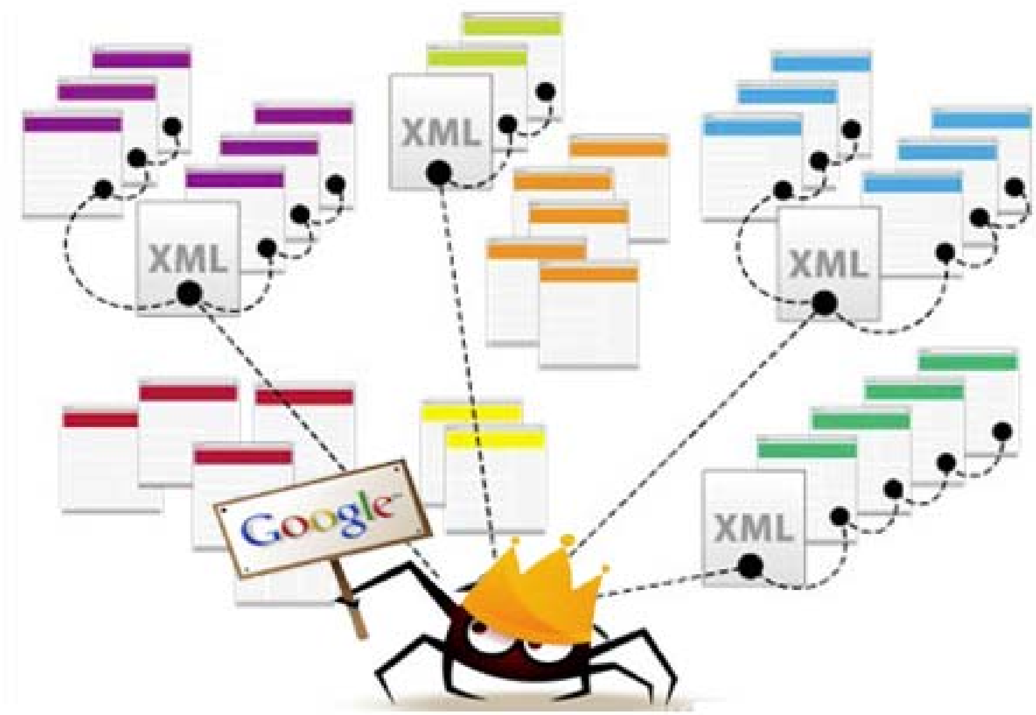
Scene completion problem



10 nearest neighbors from a collection of **2 million** images

Other similarity search applications

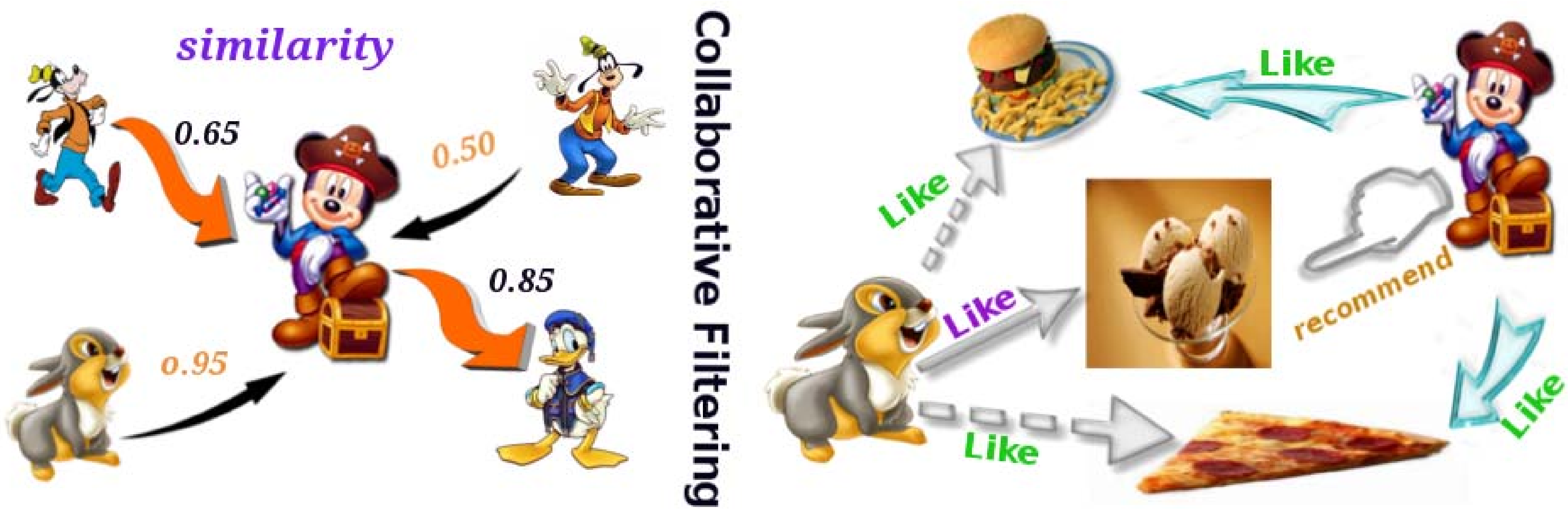
- Web crawlers and near-duplicate web pages (mirror pages)



Source: https://medium.com/@promptcloud/data-crawling-real-time-bidding-concept-mechanism-807ad8c0e0c7

Other similarity search applications

- Collaborative-filtering recommender systems require similar users or items.

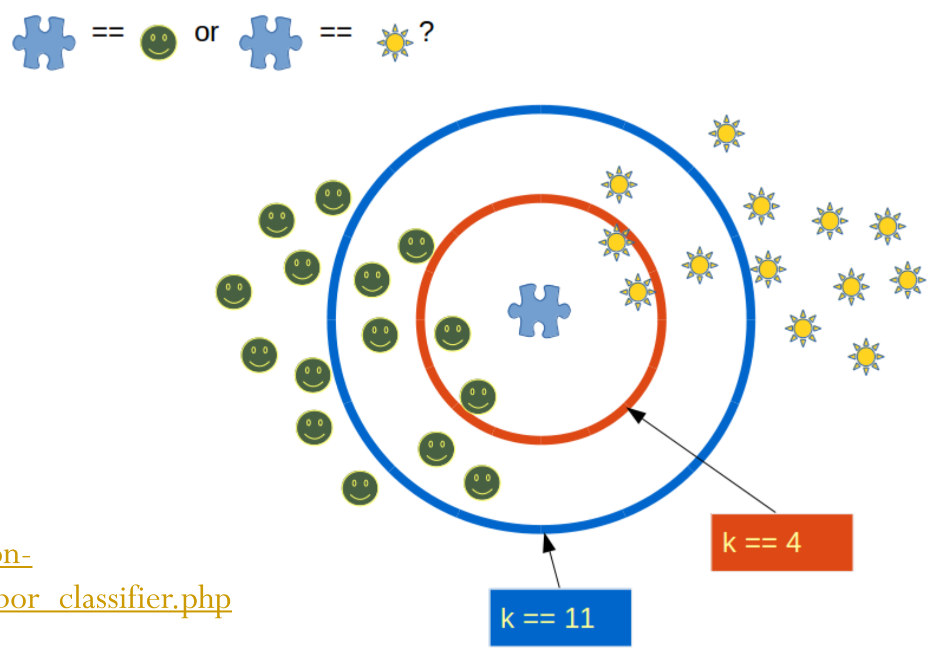


Source: https://dataaspirant.com/2015/05/25/collaborative-filtering-recommendation-engine-implementation-in-python/

A common metaphor

- We can present many problems as finding "similar" items or finding near-neighbors in high dimensions.

- More examples in machine learning:
  - **k**-nearest neighbor models (classification/regression/outlier detection)



Source: https://www.python-course.eu/k_nearest_neighbor_classifier.php

A common metaphor

- We can present many problems as finding "similar" items or finding near-neighbors in high dimensions.

- More examples in machine learning:
  - Clustering

Source: https://data-flair.training/blogs/clustering-in-r-tutorial/

General problem for our today's lecture

- Input:
  - A data set **X** of **n** points in high-dimensional space.
  - A similarity function **sim** and a similarity threshold **s**.

- Goal:
  - Find all similar pairs $\mathbf{x}_i$, $\mathbf{x}_j$ in **X** such that $\mathbf{sim}(\mathbf{x}_i, \mathbf{x}_j) \geq \mathbf{s}$.

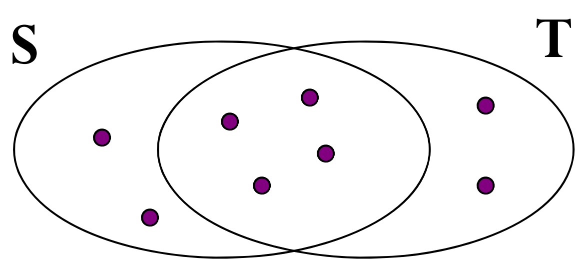
- Solutions:
  - Naïve solution: $\mathbf{O(n^2)}$.
  - Magic: This can be done in $\mathbf{O(n)}$?

Jaccard similarity

- The Jaccard similarity of two sets **S** and **T** is the ratio of their intersection size to their union size.

$$J(S,T) = \frac{|S \cap T|}{|S \cup T|}$$

- Example:



4 in intersection
8 in union
J(S,T) = 1/2.

Finding similar documents

- Input:
  - **n** documents (e.g. web pages).
  - The Jaccard similarity function **J** and a similarity threshold **s**.
- Goal:
  - Find all near-duplicate pairs **S,T** such that $\mathbf{J(S,T) \geq s}$.
- Problems:
  - Many small pieces of a document (e.g. words, phrases…) can appear out of order in another.
  - **n** documents are too large to fit into main memory.
  - Algorithmic challenge: $\mathbf{O(n^2)}$ pairs of documents to compare.

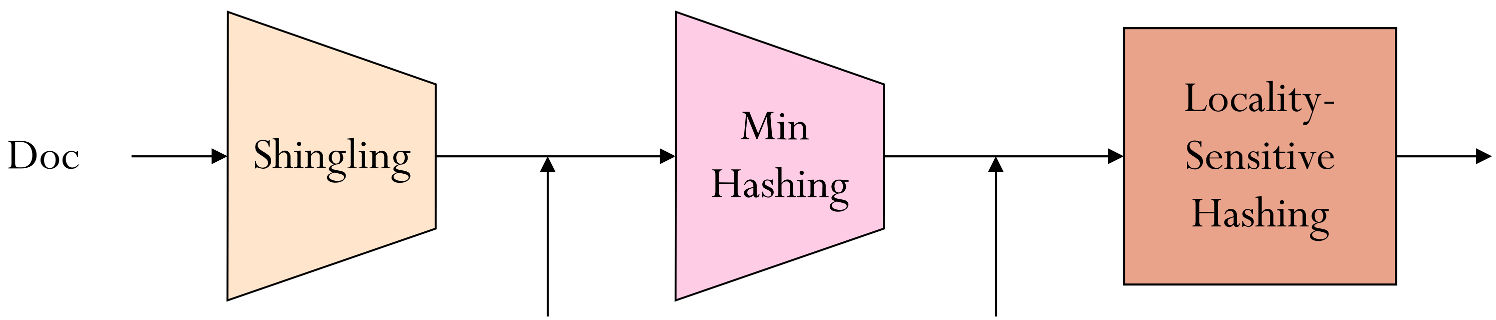
Outline

- Similarity search applications
- Finding similar documents:
  - Shingling
  - MinHash
  - Locality-sensitive hashing (LSH)

3 essential steps

- Shingling:
  - To convert documents to sets.
- MinHash:
  - To convert large sets to short signatures, while preserving Jaccard similarity.
- Locality-sensitive hashing (LSH):
  - To focus on candidate pairs which are likely from similar documents to break the barrier of **$O(n^2)$.**

Framework picture

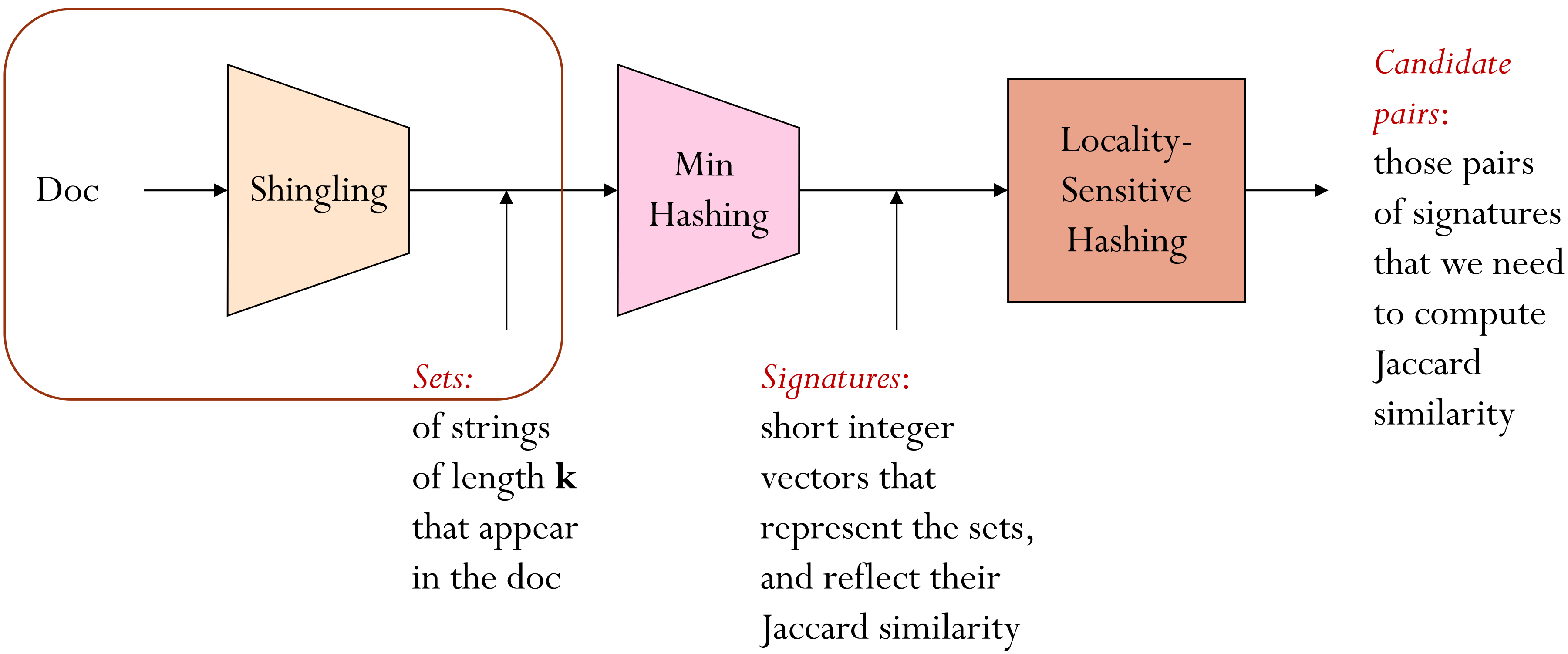


Candidate pairs: those pairs of signatures that we need to compute Jaccard similarity

Sets: of strings of length **k** that appear in the doc

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

Framework picture



Documents as high-dimensional data

- Motivation:
  - A: *A rose is red, a rose is white.*
  - B: *A rose is white, a rose is red.*
  - C: *A rose is a rose is a rose.*

- Simple approaches:
  - Document = set of words in document.
  - Document = set of "important" words in document.
  - Not work well.

- Take into account the order of words.

Shingling: Convert documents to sets

- **k**-shingle is a sequence of **k** tokens (e.g. characters, words).
- Document = set of **k**-shingles

- Example with **k** = 3:
  - A: *A rose is red, a rose is white.*
  - *A' = {a rose is, rose is red, is red a, red a rose, rose is white}.*
  - B: *A rose is white, a rose is red.*
  - *B' = {a rose is, rose is white, is white a, white a rose, rose is red}.*
  - C: *A rose is a rose is a rose.*
  - *C' = {a rose is, rose is a, is a rose}.*

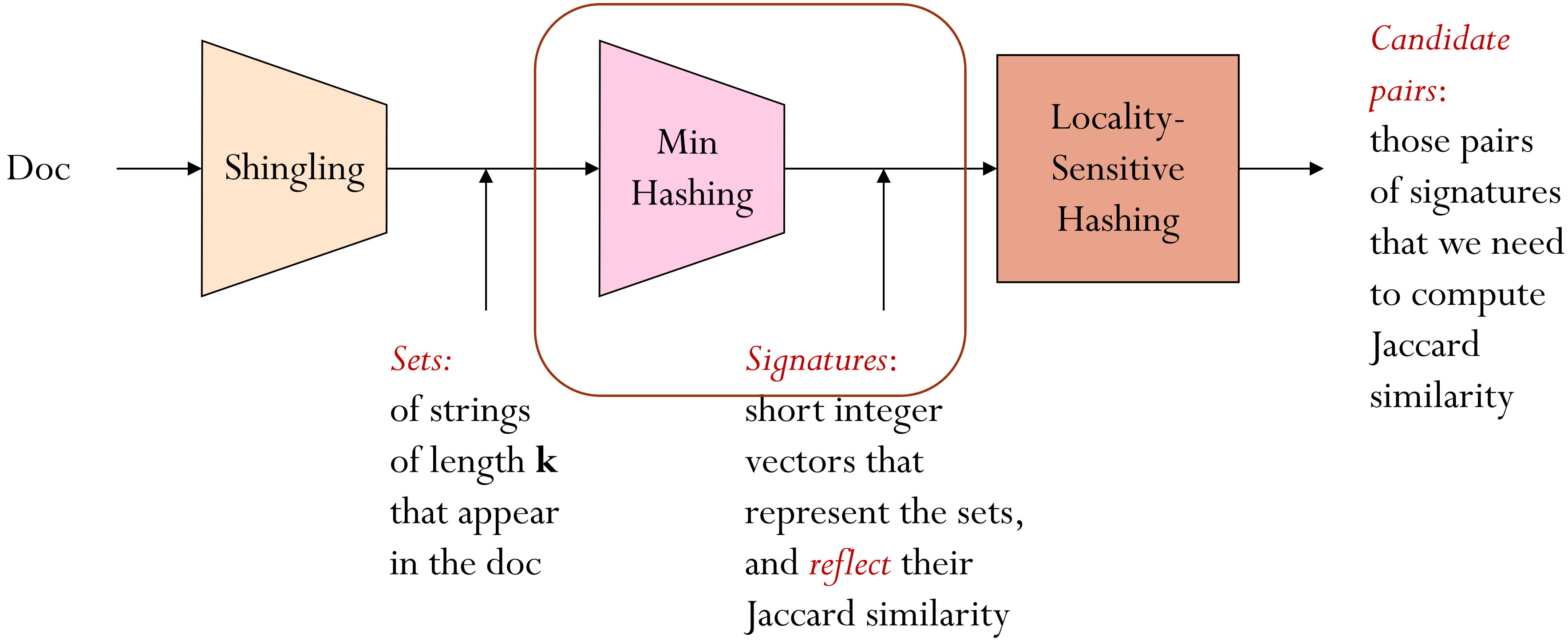
Compressing shingles

- To compress long shingles, we can hash them to **4-byte** integer.
- Represent a document as a set of hash values (integers) of its **k**-shingles.
- Example with **k** = 3:
  - $A' = \{$*a rose is, rose is red, is red a, red a rose, rose is white*$\}$.
  - $A' = \{1, 3, 8, 5, 7\}$.
  - $B' = \{$*a rose is, rose is white, is white a, white a rose, rose is red*$\}$.
  - $B' = \{1, 7, 9, 2, 3\}$.
  - $C' = \{$*a rose is, rose is a, is a rose*$\}$.
  - $C' = \{1, 4, 6\}$.

Working assumption

- If documents are similar, they will share many common shingles, hence, will have a high Jaccard similarity.
- We should pick **k** large enough to differentiate documents
  - **k = 5** for short documents.
  - **k = 10** for long documents.

Framework picture



Encoding a set as bit vectors

- Encoding for ease of presenting MinHash idea:
  - Encode a set by a binary vector.
  - Each dimension is an element in the universal set.

- Examples:
  - **A** = {1, 2, 5, 6, 7}
  - **B** = {1, 2, 3, 6}
  - **C** = {1, 6, 7}

  - Universal set: **U** = {1, 2, 3, 4, 5, 6, 7}
  - **A** = 1100111
  - **B** = 1110010
  - **C** = 1000011

From sets to a boolean matrix

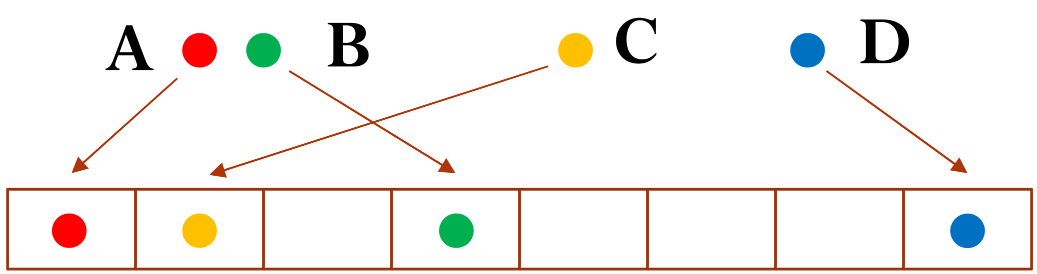
$\mathbf{A} = \{1, 2, 5, 6, 7\}$
$\mathbf{B} = \{1, 2, 3, 6\}$
$\mathbf{C} = \{1, 6, 7\}$
$\mathbf{D} = \{2, 3, 4, 5\}$

Documents / Shingles

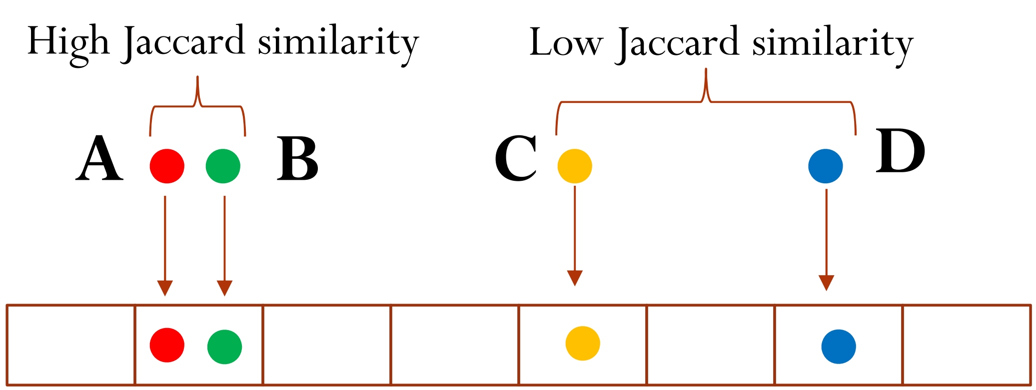
	A	B	C	D
1	1	1	1	0
2	1	1	0	1
3	0	1	0	1
4	0	0	0	1
5	1	0	0	1
6	1	1	1	0
7	1	0	1	0

Min-Hashing vs. general hashing

- General hashing:

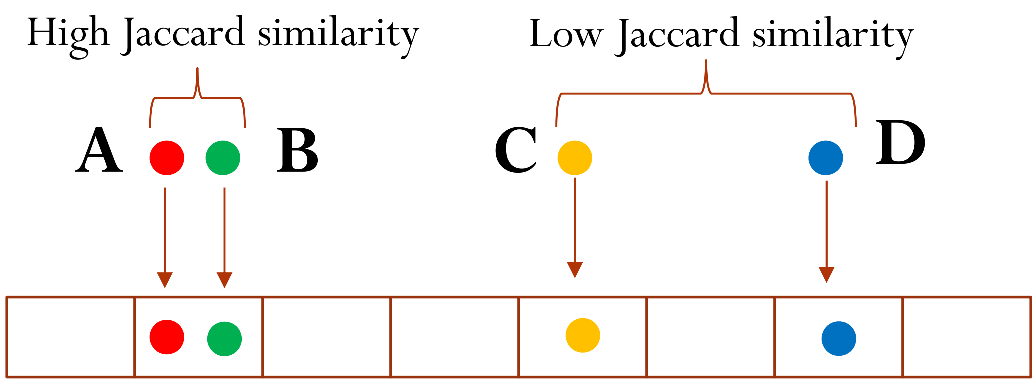


- Min-hashing:



MinHash's idea

- MinHash:



- Algorithm's idea:
  - Use MinHash to hash each set/document into a hash table.
  - Only compute Jaccard similarity of any pair of sets in the same bucket.

MinHash

- Goal: find a hash function **h** such that
  - If **J(S,T)** is high, then with high probability **h(S) = h(T)** (collision).
  - If **J(S,T)** is low, then with high probability **h(S) ≠ h(T)** (no collision) .

- MinHash for the Jaccard similarity:

$$\mathbf{Pr[h(S) = h(T)] = J(S,T)}$$

MinHash's signature

- Permute the Boolean matrix with a random permutation **π.**

- MinHash function:

***$h_\pi(S)$** = the index of the first row with value 1 of **S** (in the permuted order π)*

$$h_\pi(S) = \min_\pi \pi(S)$$

- Use several independent MinHash (i.e. permutation **π**) to construct the signature of each set.

	A	B	C	D
1	1	1	1	0
2	1	1	0	1
3	0	1	0	1
4	0	0	0	1
5	1	0	0	1
6	1	1	1	0
7	1	0	1	0

Shingles

Documents

Example

π		A	B	C	D
4	1	1	1	1	0
2	2	1	1	0	1
7	3	0	1	0	1
1	4	0	0	0	1
6	5	1	0	0	1
3	6	1	1	1	0
5	7	1	0	1	0

Shingles

Documents

Signature matrix M

A	B	C	D

Example

π	Shingles	A	B	C	D
4	1	1	1	1	0
2	2	1	1	0	1
7	3	0	1	0	1
1	4	0	0	0	1
6	5	1	0	0	1
3	6	1	1	1	0
5	7	1	0	1	0

Documents

Signature matrix M

A	B	C	D

Example

π		A	B	C	D
4	1	1	1	1	0
2	2	1	1	0	1
7	3	0	1	0	1
1	4	0	0	0	1
6	5	1	0	0	1
3	6	1	1	1	0
5	7	1	0	1	0

Shingles

Documents

Signature matrix M

A	B	C	D
			1

Example

π		A	B	C	D
4	1	1	1	1	0
2	2	1	1	0	1
7	3	0	1	0	1
1	4	0	0	0	1
6	5	1	0	0	1
3	6	1	1	1	0
5	7	1	0	1	0

Shingles

Documents

Signature matrix M

A	B	C	D
			1

Example

π	Shingles	A	B	C	D
4	1	1	1	1	0
2	2	1	1	0	1
7	3	0	1	0	1
1	4	0	0	0	1
6	5	1	0	0	1
3	6	1	1	1	0
5	7	1	0	1	0

Documents

Signature matrix M

A	B	C	D
2	2		1

Example

π		A	B	C	D
4	1	1	1	1	0
2	2	1	1	0	1
7	3	0	1	0	1
1	4	0	0	0	1
6	5	1	0	0	1
3	6	1	1	1	0
5	7	1	0	1	0

Shingles

Documents

Signature matrix M

A	B	C	D
2	2	3	1

Example

π	Shingles	A	B	C	D
4	1	1	1	1	0
2	2	1	1	0	1
7	3	0	1	0	1
1	4	0	0	0	1
6	5	1	0	0	1
3	6	1	1	1	0
5	7	1	0	1	0

Documents

Signature matrix M

A	B	C	D
2	2	3	1

3rd element of the permutation is the first to map to a 1.

Example

	π			A	B	C	D
2	3	4	1	1	1	1	0
3	4	2	2	1	1	0	1
7	7	7	3	0	1	0	1
6	2	1	4	0	0	0	1
1	6	6	5	1	0	0	1
5	1	3	6	1	1	1	0
4	5	5	7	1	0	1	0

Shingles

Documents

Signature matrix M

A	B	C	D
2	2	3	1

Example

π

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

Shingles	A	B	C	D
1	1	1	1	0
2	1	1	0	1
3	0	1	0	1
4	0	0	0	1
5	1	0	0	1
6	1	1	1	0
7	1	0	1	0

Documents

Signature matrix M

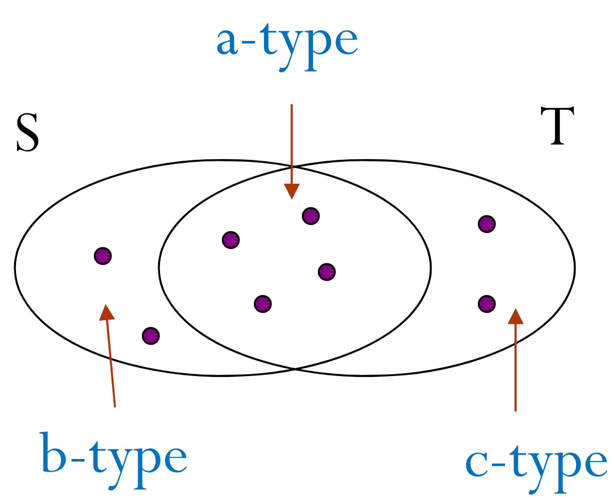
A	B	C	D
2	2	3	1
1	1	1	2
1	2	2	1

MinHash property

- Claim: $\mathbf{Pr_{\pi}[h_{\pi}(S) = h_{\pi}(T)] = J(S,T)}$

- Observation after permutation:

	S	T
a-type	1	1
b-type	1	0
c-type	0	1
d-type	0	0



- Look down the column,
  - If we observe a-type: $\mathbf{h_{\pi}(S) = h_{\pi}(T)}$
  - If we observe b-type or c-type: $\mathbf{h_{\pi}(S) \neq h_{\pi}(T)}$
  - $\mathbf{Pr[h_{\pi}(S) = h_{\pi}(T)]}$ = (a-type) / (a-type + b-type + c-type) = $\mathbf{J(S,T)}$

Similarity of signatures

- MinHash property:

$$\mathbf{Pr_\pi[h_\pi(S) = h_\pi(T)] = J(S,T)}$$

  - We use **d** MinHash functions (i.e. random permutations) to achieve a signature of size **d** for each set.
  - The similarity of two signatures = the fraction of hash functions in which they agree (the number of collisions / **d**).

- The Jaccard similarity of two sets (columns) equals to the expected similarity of their signatures.

Example

π

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

Shingles	A	B	C	D
1	1	1	1	0
2	1	1	0	1
3	0	1	0	1
4	0	0	0	1
5	1	0	0	1
6	1	1	1	0
7	1	0	1	0

Documents

Signature matrix M

A	B	C	D
2	2	3	1
1	1	1	2
1	2	2	1

Similarities:

	A-B	C-D
Col/Col	0.5	0
Sig/Sig	0.67	0

MinHash signatures

- Pick $\mathbf{d} = \mathbf{100}$ random permutations, we can represent a document as a short integer vector.
  - The corresponding column of the signature matrix **M**.
  - Preserve the pairwise Jaccard similarity.
  - The size of signature is very small, i.e. ~100 bytes.
- Important note of implementation:
  - Permute rows even once is very expensive.
  - One-pass implementation (reading 3.3.5, chapter 3 of Mining of Massive Datasets).

One-pass MinHash signatures

- Idea: For each column **S** and a universal hash function $\mathbf{h_i}$, keep a "slot" for the min-hash value.

- Algorithm:
  - Initialize all **sig(S)[i]** = ∞.
  - Scan rows looking for **1s**.
    - Suppose row **j** has **1** in column **S**.
    - Then for each $\mathbf{h_i}$:
      - If $h_i(j)$ < **sig(S)[i]**, then **sig(S)[i]** ← $h_i(j)$.

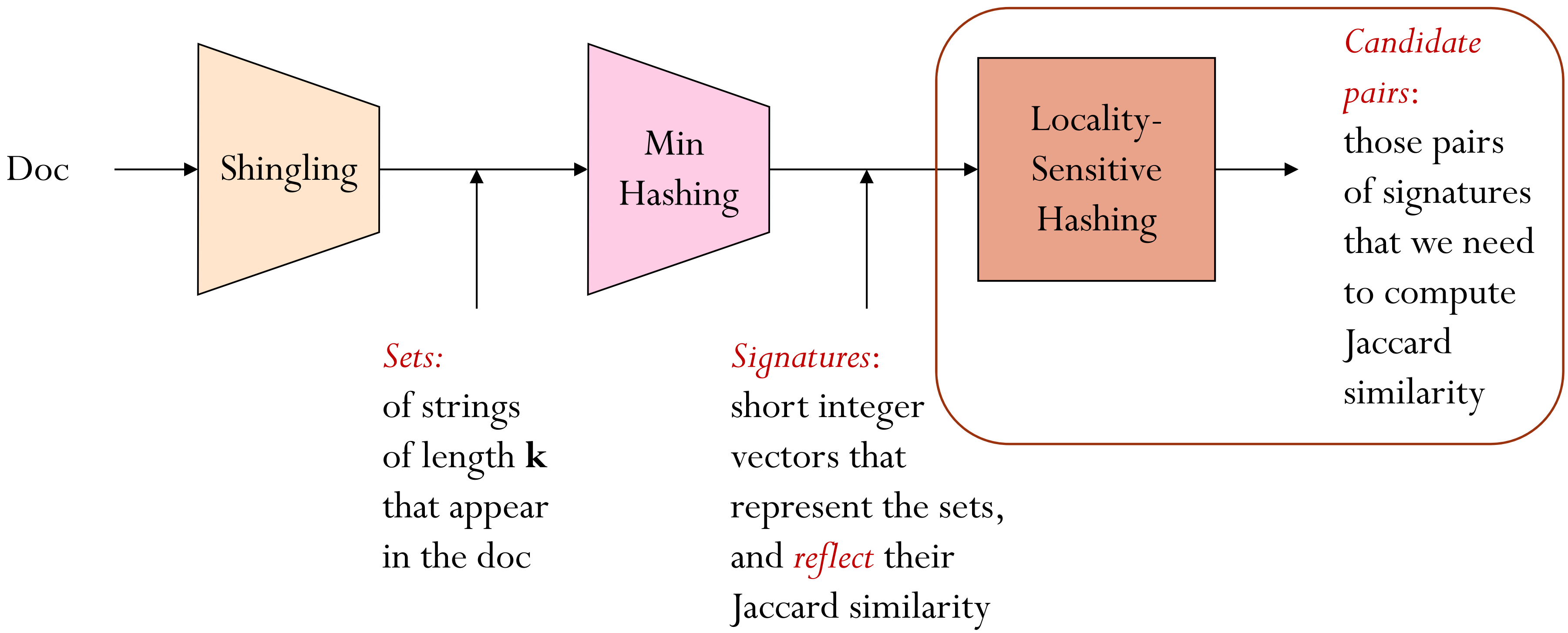
- How to pick a universal hash function **h(x)**?
  - $\mathbf{h_{a,b}(x)=((a*x+b) \bmod p) \bmod n}$ where $\mathbf{0 \leq a, b < p}$ and $\mathbf{p}$ is a large prime $> \mathbf{n}$.

Exercise (section 3.3.5)

- Compute MinHash signatures using the two hash function $\mathbf{h_1(x) = x + 1 \bmod 5}$ and $\mathbf{h_2(x) = 3x + 1 \bmod 5}$.

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

Framework picture



Motivation of MinHash/LSH

- **n = 1 million** documents.
- A naïve solution needs to compute $\mathbf{n(n-1) / 2 \sim 5*10^{11}}$ Jaccard similarities.
- A standard PC computes $\mathbf{10^6}$ Jaccard similarities/sec, then we need approximate **5 days**.

- Nowadays, the internet has **n ≈ 4.5 billion** pages ☹.

LSH: First cut

- Input:
  - The MinHash signature matrix **M.**

- Goal:
  - Find all pairs of documents with Jaccard similarity $\geq \mathbf{s = 0.8}$.

- LSH idea:
  - The higher Jaccard similarity, the larger fractions of identical hash values.
  - A pair of signatures is a candidate (which needs to compute the actual Jaccard similarity) if their hash values are the same on at least **r** fractions.

- Problem:
  - Choose **r** such that we do not have many false negatives (pairs with $\mathbf{J \geq 0.8}$ but not candidates) and false positives (pairs with $\mathbf{J < 0.8}$ but candidates).

Naïve solution and analysis

- **r = 1**:
  - **J(S,T) = 0.8** (should be retrieved)
    - Probability that **(S,T)** is a candidate: **0.8**
    - Probability that **(S,T)** is not a candidate: **1 – 0.8 = 0.2**
  - **J(S',T') = 0.5** (should not be retrieved)
    - Probability that **(S',T')** is a candidate: **0.5**
    - Probability that **(S',T')** is not a candidate: **1 – 0.5 = 0.5**

False negatives

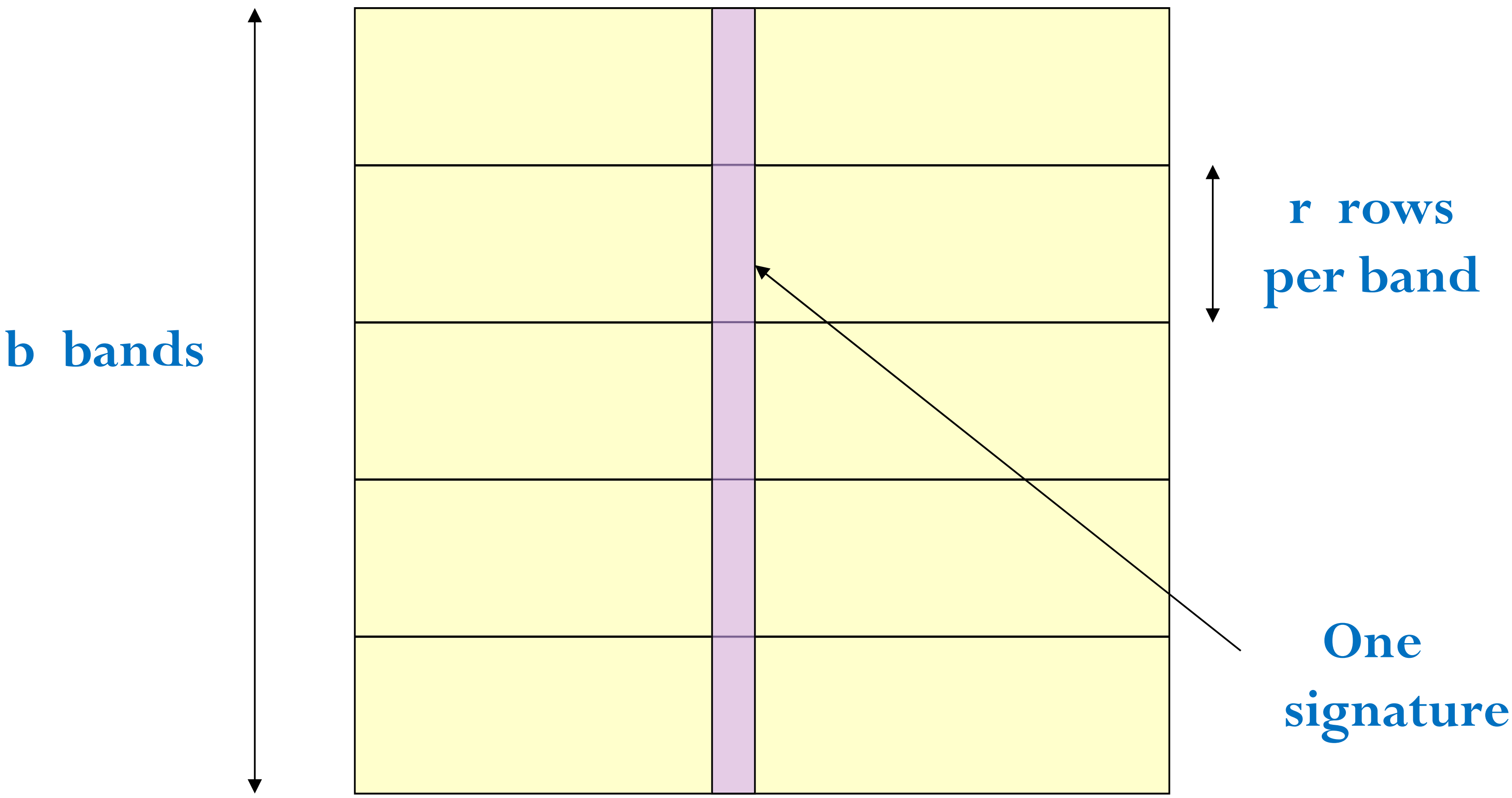
False positives

- **r = 5:**
  - **J(S,T) = 0.8** (should be retrieved)
    - Probability that **(S,T)** is a candidate: $\mathbf{0.8^5 = 0.328}$
    - Probability that **(S,T)** is not a candidate: $\mathbf{1 - 0.8^5 = 0.672}$
  - **J(S',T') = 0.5** (should not be retrieved)
    - Probability that **(S',T')** is a candidate: $\mathbf{0.5^5 = 0.031}$
    - Probability that **(S',T')** is not a candidate: $\mathbf{1 - 0.5^5 = 0.969}$

False negatives

False positives

Partition M into b bands

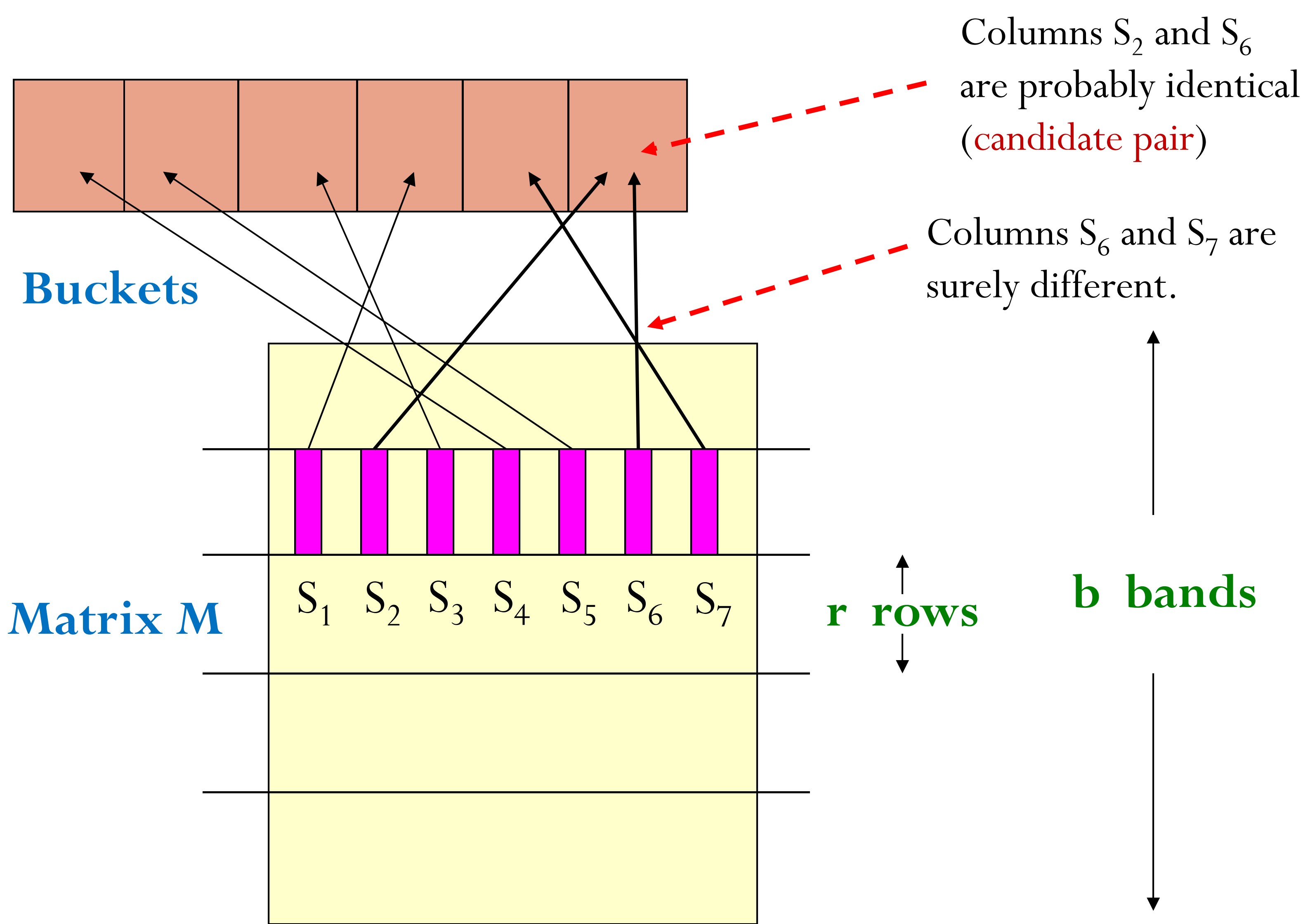


Signature matrix M

Partition M into b bands

- **M** contains **b** bands, each band has **r** rows.
- For each band, hash its **r** rows of each column into a hash table (using general hash functions) with very large buckets.
  - The collision probability of two different rows is super tiny.
- For each band, candidates are column pairs those hash to the same bucket.
  - Candidate pairs might be on the same bucket for more than **1** band.
- Tune **b** and **r** to return most similar pairs but few dissimilar pairs.

Partition M into b bands



Simplifying assumption

- The number of buckets is sufficiently large so that two different columns will hash into different buckets.
- Hence, the "same bucket" means "identical in that band".
- The assumption will simplify analysis, not for correctness of algorithms.

Analysis for J(S, T) = 0.8

- Input: Divide matrix **M** of **d = 100** rows into **b = 20, r = 5.**
- Goal: **(S,T)** is a candidate, i.e. **S** and **T** hashed into the same bucket for at least 1 band.

- Analysis:
  - Probability that **(S,T)** identical in **1** band: $\mathbf{0.8^5 = 0.328}$
  - Probability that **(S,T)** not identical in **1** band: $\mathbf{1 - 0.8^5 = 0.672}$
  - Probability that **(S,T)** not identical in all **20** bands: $\mathbf{(1 - 0.8^5)^{20} = 0.00035}$
  - Probability that **(S,T)** identical in at least **1** band: $\mathbf{1 - 0.00035 = 0.99965}$

- Conclusion:
  - About **1/3000** of the **80%**-similar column pair are false negatives (not return).
  - We can find **99.965%** truly similar pairs of documents.

Analysis for J(S, T) = 0.3

- Input: Divide matrix **M** of **d = 100** rows into **b = 20, r = 5.**
- Goal: **(S,T)** is not a candidate, i.e. **S** and **T** are not hashed into the same bucket for all bands.

- Analysis:
  - Probability that **(S,T)** identical in **1** band: $\mathbf{0.3^5 = 0.00243}$
  - Probability that **(S,T)** not identical in **1** band: $\mathbf{1 - 0.3^5 = 0.99757}$
  - Probability that **(S,T)** not identical in all **20** bands: $\mathbf{(1 - 0.3^5)^{20} = 0.9525}$
  - Probability that **(S,T)** is a candidate: $\mathbf{1 - 0.9525 = 0.0475}$

- Conclusion:
  - Approximately **4.75%** of the **30%**-similar column pair are false positives (candidate pair).
  - We will not report them since we can compute their actual Jaccard similarity.

Tradeoff of LSH

- Parameter settings to balance the false negatives/positives:
  - The number of MinHash (**d**) to construct the signature matrix **M.**
  - The number of bands **b.**
  - The number of rows **r** per band.

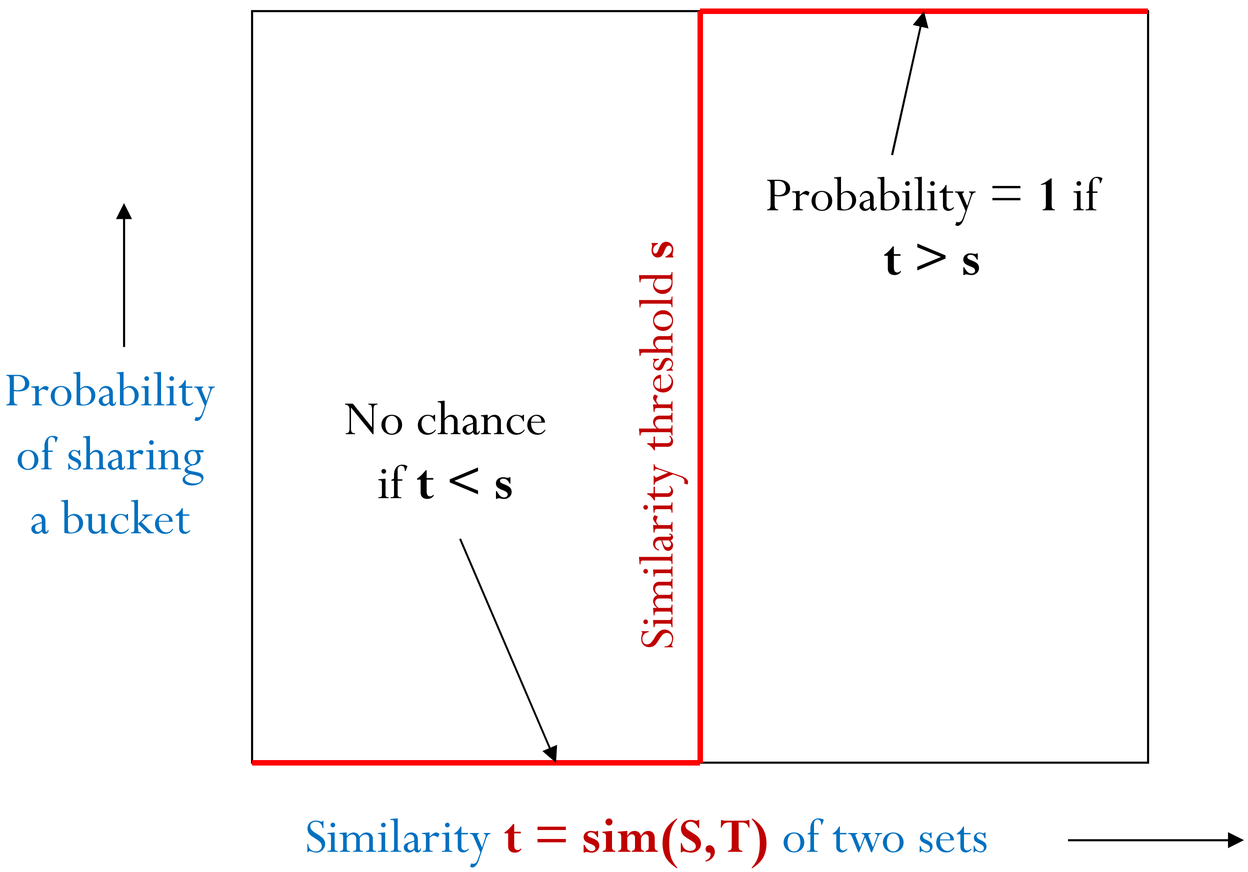
- Exercise:
  - If we set **b = 15, r = 5** rows:
    - What is the probability **80%**-similar pair is not a candidate?
    - What is the probability **30%**-similar pair is a candidate?
    - False negatives increase, false positives decrease.

General analysis for $J(S, T) = t$

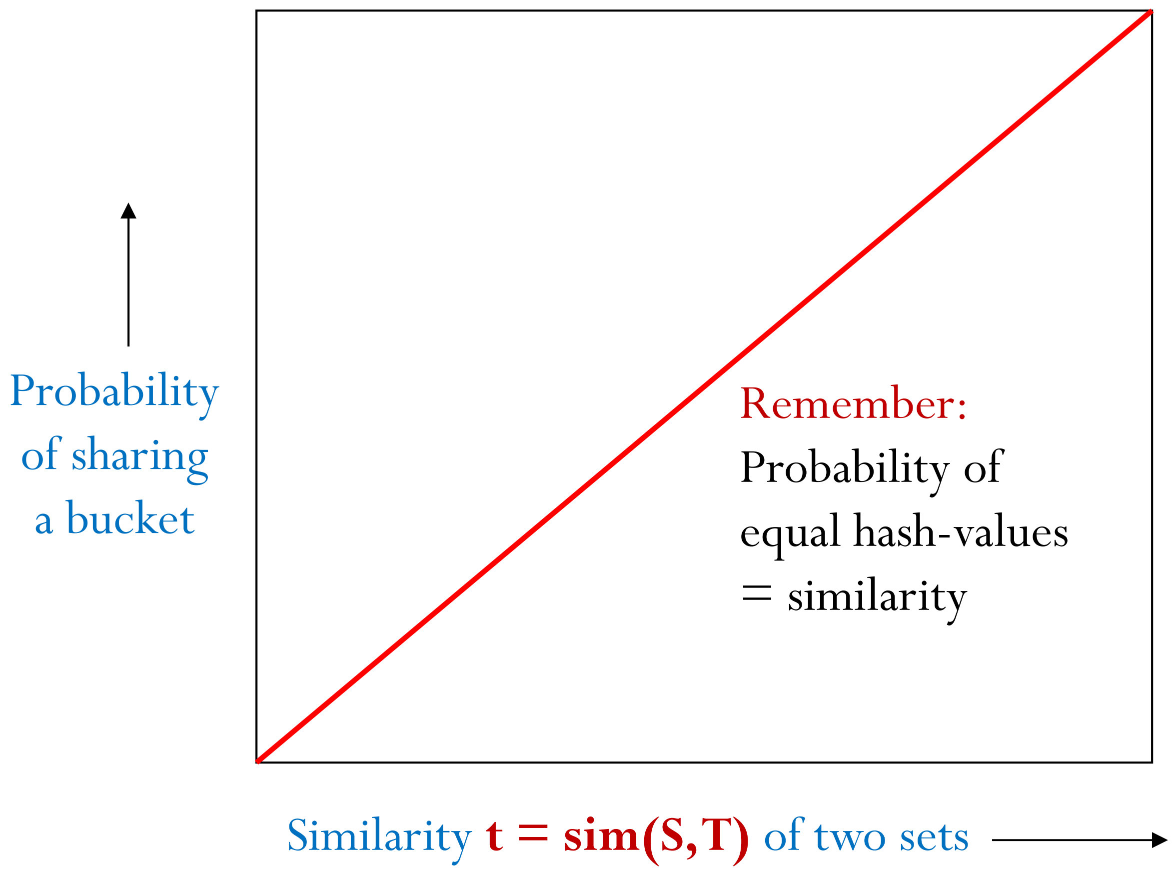
- Parameters:
  - **b** bands and **r** rows per band.

- Analysis:
  - Probability that **(S,T)** identical in **1** band: t^r
  - Probability that **(S,T)** not identical in **1** band: $1 - t^r$
  - Probability that **(S,T)** not identical in all **b** bands: $(1 - t^r)^b$
  - Probability that **(S,T)** is not a candidate: $(1 - t^r)^b$
  - Probability that **(S,T)** is a candidate: $1 - (1 - t^r)^b$

Ideal case – what we want

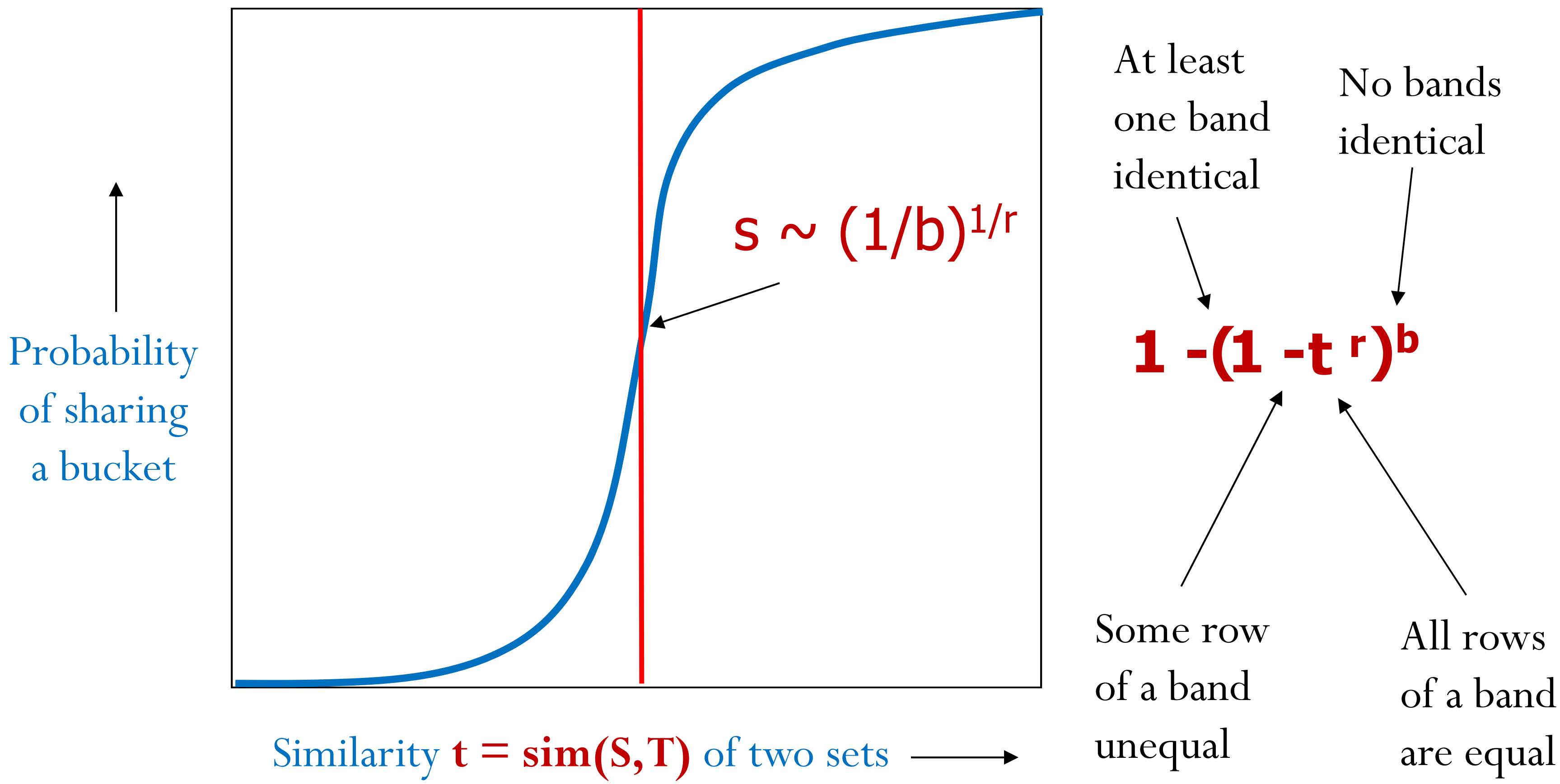


b = 1, r = 1



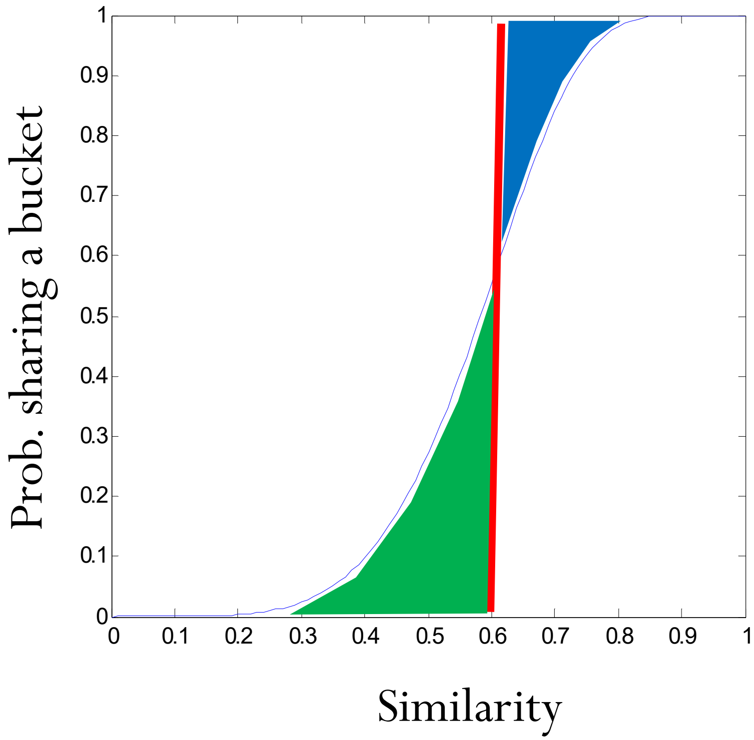
Probability
of sharing
a bucket
Remember:
Probability of
equal hash-values
= similarity
Similarity **t = sim(S,T)** of two sets

General b and r



General b and r form the S-curse

- Pick **r** and **b** to get the best S-curse.
- Example: **b = 10, r = 5, d = 50**



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

LSH summary

- Tune **b** and **r** such that
  - Decrease false negatives: get almost all pairs with similar signatures.
  - Decrease false positives: eliminate most pairs that do not have similar signatures.
- Check in main memory that candidate pairs really do have similar signatures (i.e. estimate Jaccard similarity)
- Optional: In another pass through data, compute the actual Jaccard similarity of candidate pairs to return the result.

Summary of 3 steps

- Shingling: Convert documents to sets
  - We used hashing to assign each shingle an integer ID.

- Min-Hashing: Convert large sets to short signatures, while preserving Jaccard similarity.
  - We used similarity preserving hashing to generate signatures with property $\mathbf{Pr_{\pi}[h_{\pi}(S) = h_{\pi}(T)] = J(S,T)}$.
  - We used hashing to get around generating random permutations (efficient implementation).

- Locality-Sensitive Hashing: Focus on pairs of signatures likely to be from similar documents.
  - We used hashing to find candidate pairs of Jaccard similarity $\geq$ **s**.