Analyzing Massive Data Sets

Summer Semester 2019

Prof. Dr. Peter Fischer

Institut für Informatik

Lehrstuhl für Datenbanken und Informationssysteme

Chapter 4: Finding Similar Items

High-Dimensional Data and Similarity

- First *conceptual* and *algorithmic* part of the lecture
- Two core concepts:
  - **High-Dimensional Data**: Data items represented by many data points (hundreds, thousands, … possibly out of a much large space)
  - Analyzing a *single* or *few* dimensions insufficient to understand items
  - **Similarity/Distance**: Expressing pair-wise similarity over all features

- Applications:
  - **Finding Similar Items**: pairwise (this chapter)
  - **Clustering**: Identify structure / groups using similarity
  - **Retrieval**: Similarity between search expression and data set

- Strategies for massive volumes:
  - **Exact solutions** are **costly** – but there are several strategies to help
  - **Approximate solutions** more **feasible** – e.g., multiple hashes

Similar Items - A Common Metaphor

- **Many problems can be expressed as finding "similar" sets:**
  - **Find *near-neighbors* in *high-dimensional* space**
- **Examples:**
  - **Pages with similar words (around 1.7% - 7% pages on the web)**
    - Mirror pages
    - Common source pages
    - Plagiarism
    - Classification by topic
  - **Customers who purchased similar products**
    - Foundation of recommendation systems (think Amazon)
    - Products with similar customer sets
  - **Media with similar content (images, music, videos)**
    - Duplicate removal
    - Recommendations

Defining Similarity

- Representing the data items:
  *What is **specific** about items in a collections?*
  - Text: Character Frequency, Lexical, Structure (sentence, chapters), Semantics/Meaning
  - Images: Colors, Structure, Objects, …
  - Music: Pitch, Melody, Metric/Rhythm, Modulation, …

- Expressing Similarity:
  *Given the features, what makes items **close**?*
  - Shared features
  - Numerically similar features values
  - Relevance of certain features
  - Combination of features

Computing Similarity

- **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
- Documents are so large or so many that they cannot fit in main memory

- **An exact solution is possible in $O(N \log N)$**
- **An approximate version can be done in $O(N)$!!**
- **How?**

Easier problem: Identical copies

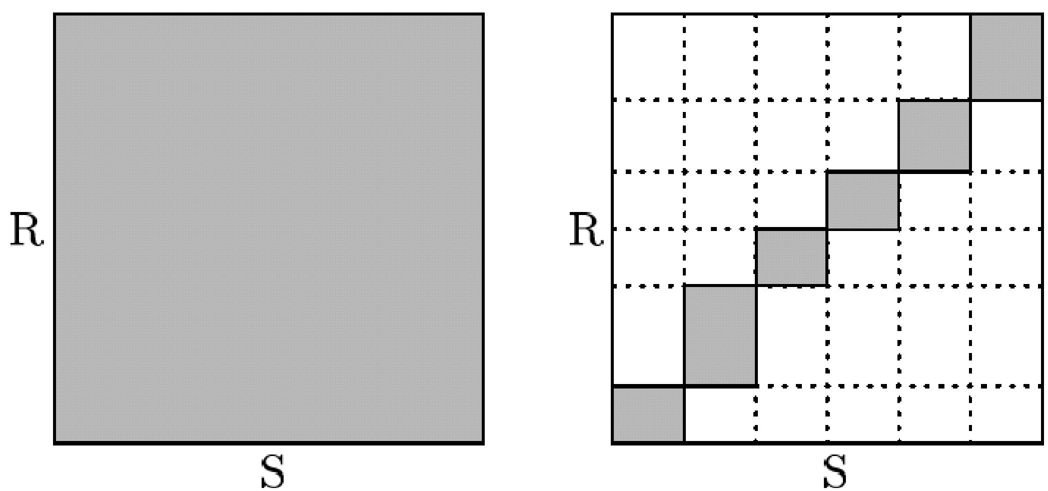
- Relatively straightforward task
- Naive strategy:
  - Enumerate all pairs $\frac{n^2}{2}$ -> $O(n^2)$
  - Do a bitwise comparison, e.g. `cmp`: True or False

- Can be solved with cost O(n)
- How?

- Apply a hash function on every element
- Elements with same content will be in the same bucket
- Caveat: Other direction may not hold:
  same hash value does not imply same content
- -> Need to check for collisions
- Scales well: hashing can be done fully parallel
- Hash tables can be distributed

Similar problem: Join processing

- $A \times B + \sigma_{\text{A.a comp B.b}}$
- For all pairs, apply a comparison => again $O(n^2)$
- Joins can reduce the search space for certain predicates:



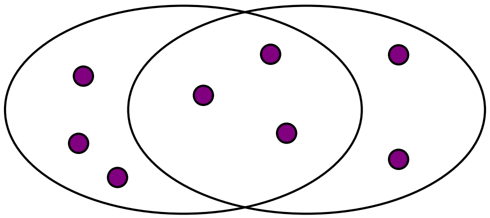
Strategies to reduce search space

- Join search space reduction
  - Indexing: Allow faster access to matching "inner" elements
  - Sorting: order one or both sides
  - Hashing: reduce space to items with same hash value
- Will not get O(n) complexity in the general case
  - Common Indexes (B+-Trees) have O(n log n) creation and access cost
  - Sorting costs O(n log n)
  - Indexes and sorting don't work well with many dimensions
  - Hashing only works for equality
- Is this a dead-end?
- No, we but can play with these concepts!

Expressing Similarity

Distance Measures (first take)

- **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
- **Common for texts: 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$_2$) =** $\frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$
  - **Jaccard distance: *d*(C$_1$, C$_2$) = 1 -** $\frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$



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

Requirements for distance functions

Two data instances/points:

- $d(x, y) \geq 0$ (no negative distances)
- $d(x, y) = 0$ if and only if $x = y$ (distances are positive, except for the distance from a point to itself)
- $d(x, y) = d(y, x)$ (distance is symmetric)
- $d(x, y) \leq d(x, z) + d(z, y)$ (the triangle inequality)

- Triangle Inequality:
  - no gain from a "detour", distance describes the shortest path
  - Hardest to prove, most often violated by candidates

"Base type" for measuresments

- Set Membership / Binary Variables
  (does the other data items contain the same features)
- Vector (Ordered Set)
- Spatial/Numeric values in Vector space
- (String) Editing: number of operations to transform data item into the other
- Graphs: common nodes and edges? Common label names?
- Time Series: sequences with timestamps, align shapes, time shifts, value shifts, …
- …

Set Membership

- Basic Idea:
  How many members of one set are also member of the other set?
- Also used for binary variables
  (0/1, True/False, Present/Absent)
- Most well-known measure - Jaccard

$$\mathrm{J}(\mathrm{C1}, \mathrm{C2}) = 1 - \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

- Many variants, not all true distance functions
  - **Sørensen–Dice:** denominator sum of sizes (~**F1 score**)
  - **Simple Matching coefficient, Rand:** also mutual absence
- Alternative: Hamming - symmetric difference

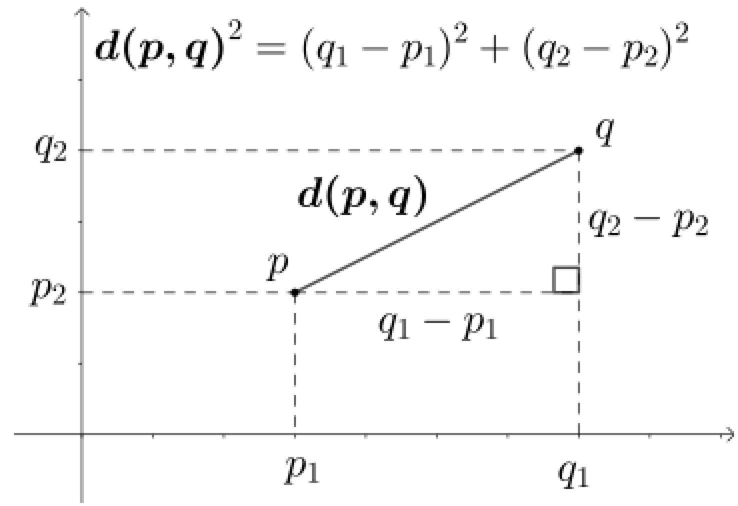
$$H(x, y) = |(x - y) \cup (\mathrm{y} - \mathrm{x})|$$

Numeric / "Spatial" Distances (1)

- Idea:
  - Observe component-wise difference
  - Scale/normalize when needed

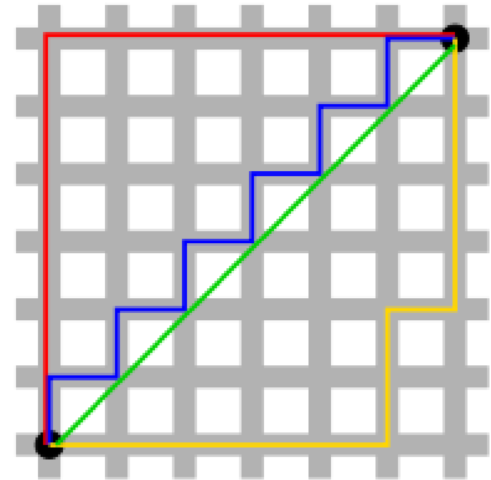
- Euclidean aka "Straight-Line":

$$E(x,y) = \sqrt[2]{\sum_{i=1}^{n}(x_i - y_i)^2}$$



- Based on Pythagorean formula for triangles: $a^2 + b^2 = c^2$

Numeric / "Spatial" Distances (2)

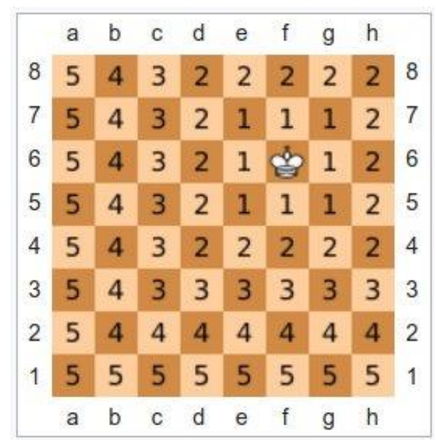


- Manhattan (aka city block, taxicab)

$$M(x,y) = \sum_{i=1}^{n} |x_i - y_i|$$

- Weighted version: Canberra

$$CB(x,y) = \sum_{i=1}^{n} \frac{|x_i - y_i|}{|x_i| + |y_i|}$$



- Chebyshev/Chessboard (maximum component):

$$C(x,y) = \ max_i(|x_i - y_i|)$$

- Observe a pattern?

Generalizing spatial metrics: P-Spaces

Minkowksi distance :

$$MK(x,y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p\right)^{\frac{1}{p}}$$

- Manhattan distance: P=1
- Euclidean distance: P=2
- Chebyshev distance: $\lim_{p\to\infty} MK(x,y)$

- Definition allows for arbitrary, even negative p

Cosine similarity/distance

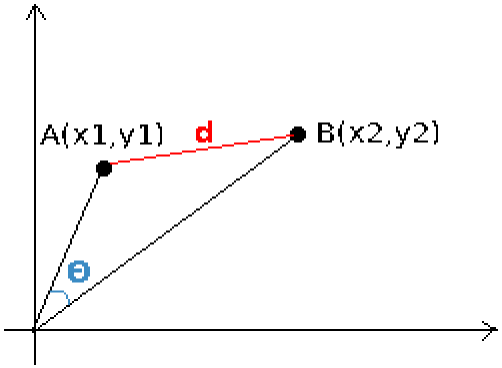
- Angle between vectors
- Vector Product, normalized for length

$$C(x,y) = 1 - \frac{x \cdot y}{||x||*||y||} = 1 - \frac{\sum_{i=1}^{n} x_i y_i}{\sqrt{\sum_{i=1}^{n} x_i^2}\sqrt{\sum_{i=1}^{n} x_i^2}}$$

- Why cosine?:

$$x \cdot y = ||x|| * ||y|| * \cos\theta$$

- Cosine vs Euclidean
  - Cosine ignores length of vectors
  - Useful for weighted data
    (e.g., frequency counts in documents)



Edit / String Distance

- Measure distance between character sequences by the number of edit operations
- kitten -> sitting
- Levenshtein: Distance 3
  - Replace s for k
  - Replace i for e
  - Insert g at end
- LCS (only insert+remove): Distance 5
  - Remove k
  - Insert s at beginning
  - Remove e
  - Insert i before n
  - Insert g add end

- Could also apply hamming distance here: A={k,e}, B={s,i,g}

Representing Documents

- Naïve, extreme representations not really helpful
  - Full document (only for identity)
  - Individual symbols/characters (maybe for language detection)
- Split document - what is the right granularity?
  - Individual words
  - Groups of words of characters (n-grams)
- Does the order matter?
- Does the word frequency matter?
- Or is there any other unit of importance?
  - Structure of document?

Common Representation of Texts: Vector Space

- Each feature (word, shingle) of a document is assigned a dimension.
- The number represents the weight is the value in the respective dimension.
  - A value space of 0/1 may just denote **presence** or **absence** of a feature
  - Another common approach is the *number of occurrences* (**term frequency**)
  - Normalization: bring weights in same range for all documents
  - Feature Weighing over all documents (IDF)
- Useful orderings: lexicographical, weights, …

Example (with lexicographical ordering)

[as, please, possible, soon, yes]

- D1: ”yes as soon as possible” — D1: [2, 0, 1, 1, 1]
- D2: ”as soon as possible please” — D2: [2, 1, 1, 1, 0]

- In real datasets:
  - Vectors are very long (many possible features), but also very sparse
  - Skewed frequency: few very common features, many rare features: stop words, "tail clipping"

Finding Similar Items – Exact Solution

First idea: Inverted Indexes

- Key idea: Similarity can be only greater than 0 if there are shared features
- Inverted Index: look up documents containing a term/features
- Documents (aka forward index)

w = [C,D, F]

z = [C,F, G]

y = [A,B, E]

x = [B, E, H]

Inverted index:

A : y

B : y, x

C : w, z

D : w

E : y, x

F : w, z

G : z

H: x

Find candidates:

for every term in source:

get other documents from index

Merge and traverse

Limitations of inverted indexes

- Document length and term skew:
  - Very common terms shared among many documents
  - Very long list for long words
- Many candidates, large set to store and test

Documents

- w = [C,D, F]
- z = [A,B, E, F, G]
- y = [A,B, C,D, E]
- x = [B, C,D, E, F]

Candidates:

- w = {y, x, z}
- z = {x, y, w}
- y = {x, z, w}
- x = {w, y, z}

Index :

A : z, y

B : z, y, x

C : w, y, x

D : w, y, x

E : z, y, x

F : w, z, x

G : z

Working on Prefixes

- Overlap: O(x,y) = $|x \cap y|$

(not a true distance function)

Given a threshold t, the following hold:

1. $J(x,y) \geq t \Leftrightarrow O(x,y) \geq \frac{t}{1+t} * (|x| + |y|)$
2. $J(x,y) \geq t \Rightarrow O(x,y) \geq t * |x|$
3. $J(x,y) \geq t \Rightarrow t * |x| < |y|$

- With 1 and/or 2, we can approximate J using O

Intuition for prefixes:

- for large data sets, we will aim for high thresholds
- high thresholds mean significant overlap
- With high overlap, we can eliminate candidates after seeing few non-matches when comparing (needs order)

Benefits:

- Only test prefix to find candidates
- Only index up to prefix length

Prefix with high similariy (t=0.9)

- w = [<u>C</u>,D, F]
- z = [<u>A</u>,B, E, F, G]
- y = [<u>A</u>,B, C,D, E]
- x = [<u>B</u>, C,D, E, F]
- Tokens in the prefixes are underlined.
- For each record, similarity has to be tested with respect to its candidates.
- candidates :
  - z = {y}
  - y = {z}
- We have to check only 1 pair which afterwards does not meet the threshold.

index :
A : z, y
B : x
C : w

Prefix with low similarity (t=0.5)

- w = [C,D, F]
- z = [A,B, E, F, G]
- y = [A,B, C,D, E]
- x = [B, C,D, E, F]
- candidates :
  - w = {y, x}
  - z = {x, y}
  - y = {x, z, w}
  - x = {w, y, z}
- We have to check 5 pairs - one less than in Example 1 without prefix filtering
- w, x and x, y pass the threshold.

index :
A : z, y
B : z, y, x
C : w, y, x
D : w, x
E : z

Formalizing prefixes

- Given an ordering over the tokens in all lists and O(x,y) > a, then the ($|x| - \alpha + 1$)-prefix of x and the ($|y| - \alpha + 1$)-prefix of y must share at least one token.
- Since information on both sides is needed, precomputation is not directly possible
- We can again approximate:
  O (x,y) >= t* |x| (out of 3 and 1)
- Applying the formulas before, we can determine the needed prefix length of a list u as

$$|u| - \lceil t * |u| \rceil + 1$$

Further improvement: positional filtering

- Consider t = 0.8 and
- y = [A,B, C,D, E]
- x = [B,C,D, E, F]
- x and y are candidates of each other for a final similarity check, however they will not pass the constraint O(x, y) ≥ 5 resulting from J(x, y) ≥ 0.8.
- Taking the position of the common token B into account the maximum possible overlap can be estimated with respect to the unseen tokens in x and y:
- 1 + min(3, 4) = 4 therefore x and y cannot pass the similarity check.

- Let w = x[i] be the i-th token in x.
- w partitions x into a left xl (w) and right partition xr (w).
- If O(x, y) ≥ α, then for every token w ∈ x ∩ y:

  O(xl (w), yl (w)) + min(|xr (w)|, |yr (w)|) ≥ α.

Approximative Solution

Hashing for Approximation

Key Ideas:

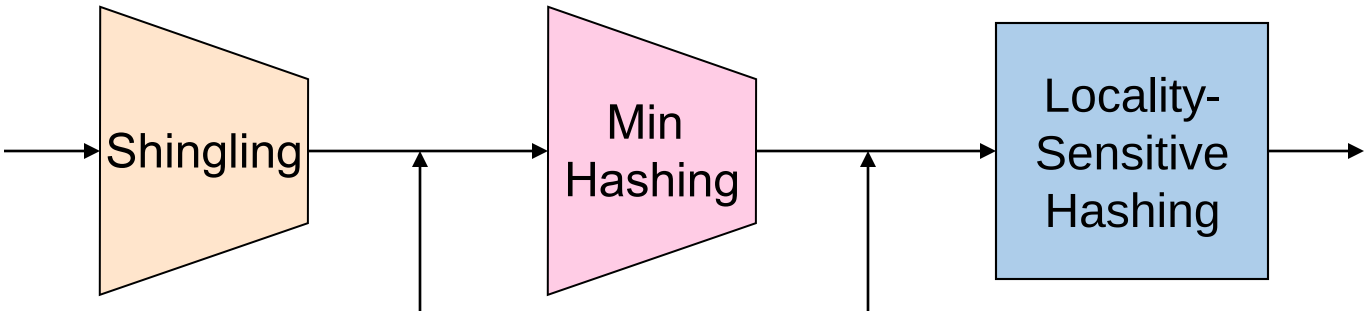
- Apply **multiple** different **hash functions** on the same data to express "more" than equality: **approximate similarity**
- Use **hash functions** that preserve of similarity (dependency on similarity function)
- Use **hashing** at **multiple stages** for **different purposes**

Strategy*:*

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
   - **Reduce search space to generate candidate pairs**
   - **Careful about false negatives**

The Big Picture

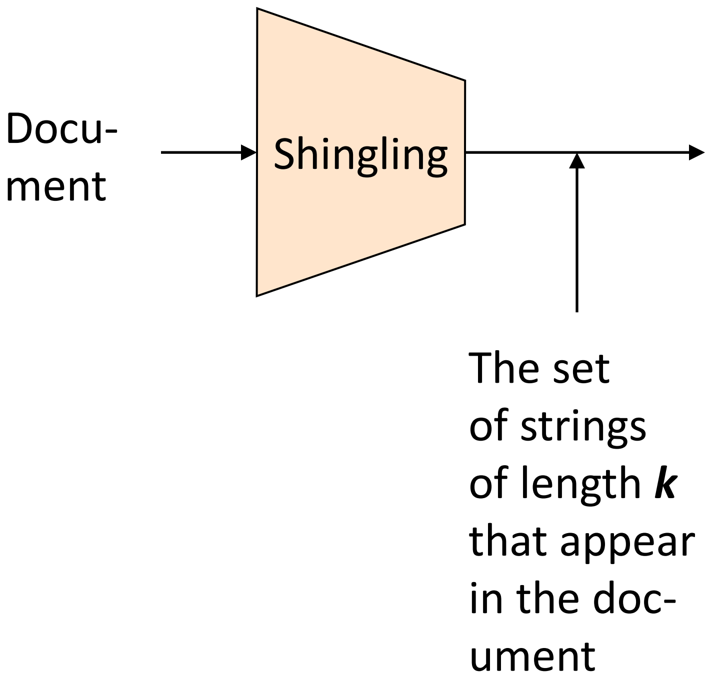
Docu-
ment



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



Shingling

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

Documents as High-Dim Data

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

- **Simple approaches:**
  - Document = set of words appearing in document
  - Document = set of “important” words
  - Don’t work well for this application. Why?

- **Need to account for ordering of words!**
- A different way: **Shingles!**

Define: Shingles

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

- **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}

Assessment of Shingles

- **Benefit:** Shingles capture some order of the document
  - Full representation of order within a shingle
  - Overlapping shingles provide indication of overall ordering
  - Reordering the document invalidates only few shingles

- **Core Tuning question: How big to make k?**
  - **Too short = most shingles in most documents**
  - **Too long = missing too many possible candidates**

Compressing Shingles

- Motivation:
  - Shingles generate a large space: token space^length, e.g., 27^9
  - Shingles consume O(length) bytes
  - String operations are (relatively) inefficient
- 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
  - In practice, the space of 2^32 is sufficient to cover all relevant shingles
- **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}

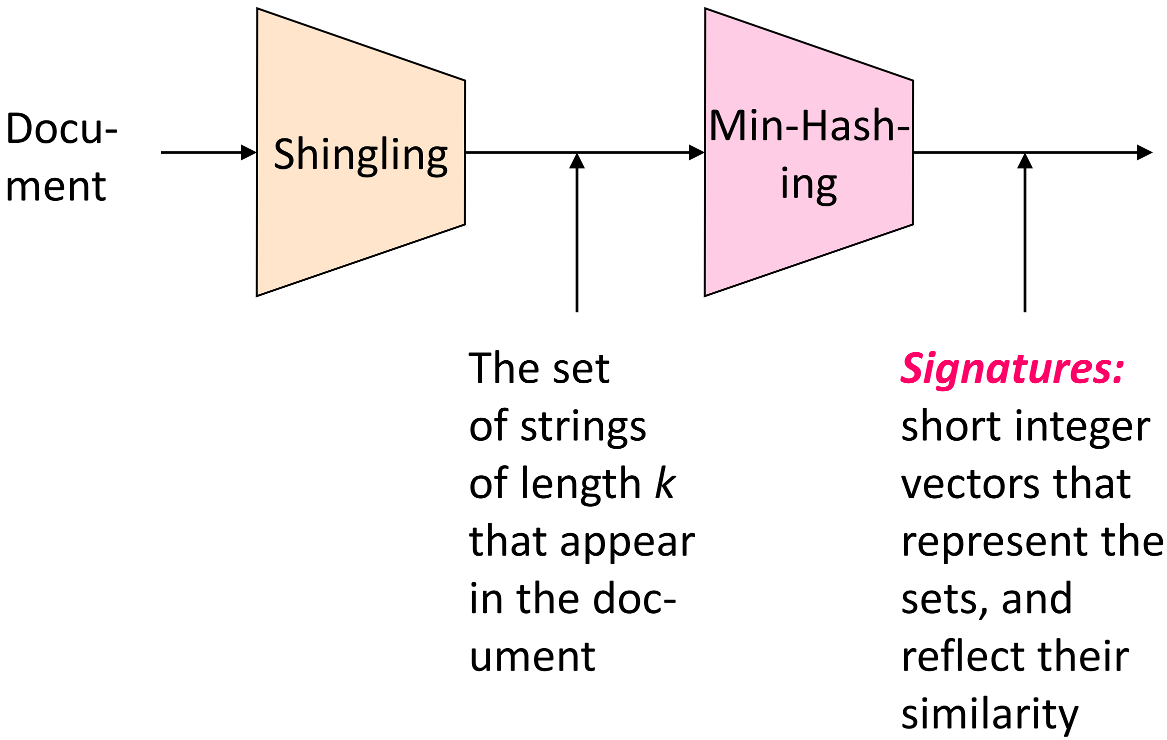
Similarity Metric for Shingles

- **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**

Assumption:

- **Documents that have lots of shingles in common have similar text, even if the text appears in different order**

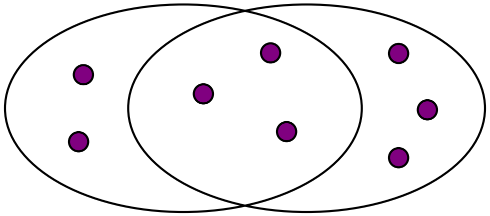
- **Caveat:** You must pick ***k*** large enough, or most documents will have most shingles
  - ***k*** = 5 is OK for short documents
  - ***k*** = 10 is better for long documents



MinHashing

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

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**

Outline: 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. $h(C_1) \neq h(C_2)$

- **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

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

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

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

0	0
0	0
1	**1**
0	0
0	1
1	0

One of the two cols had to have 1 at position ***y***

Min-Hash Signatures

- We know: **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**
- **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**
- **Real-Life Implemention: Don't permute, but use random hash functions**

Min-Hash Signature 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)

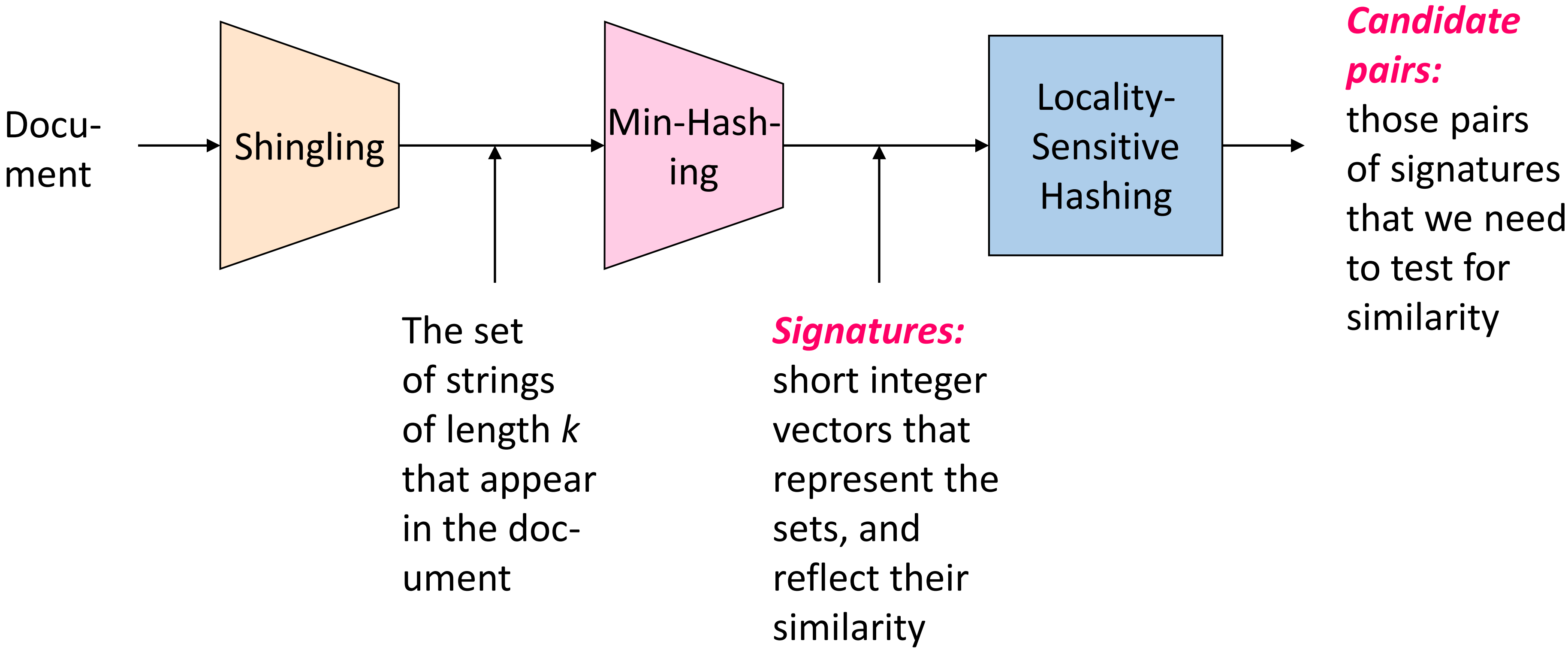
1	0	1	0
1	0	0	1
0	1	0	1
0	1	0	1
0	1	0	1
1	0	1	0
1	0	1	0

Signature matrix *M*

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

Similarities:

	1-3	2-4	1-2	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	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**
- **Caveat: This approach can generate**
  - **False positives: Same bucket but too little overlap -> check afterwards**
  - **False negatives: Never same bucket, but enough overlap**

Candidates from Min-Hash

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

- **Pick a similarity threshold *s* (0 < s < 1)**

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

LSH for Min-Hash

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

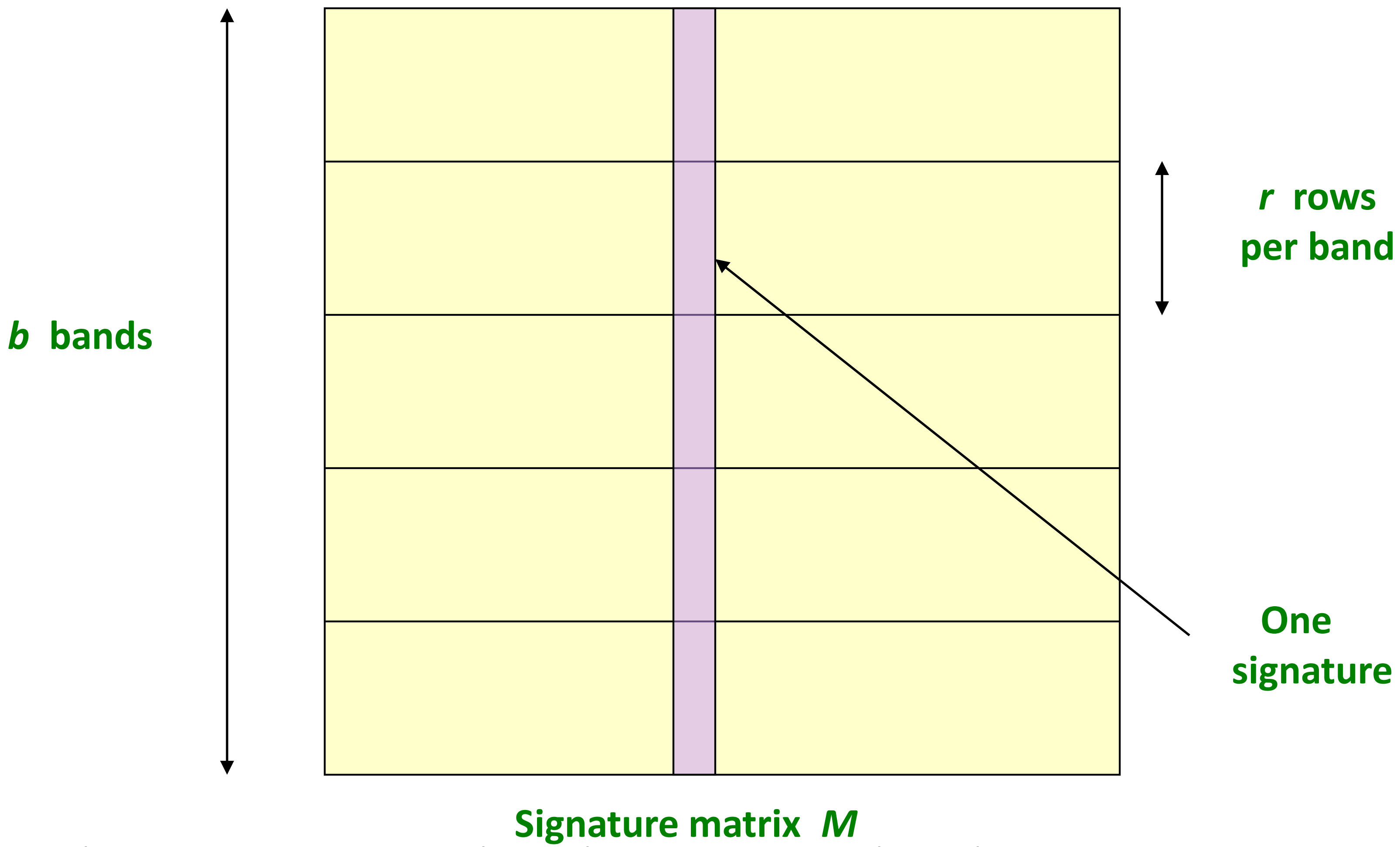
- **Big idea: Hash columns of signature matrix *M* several times**

- 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



Partition M into Bands

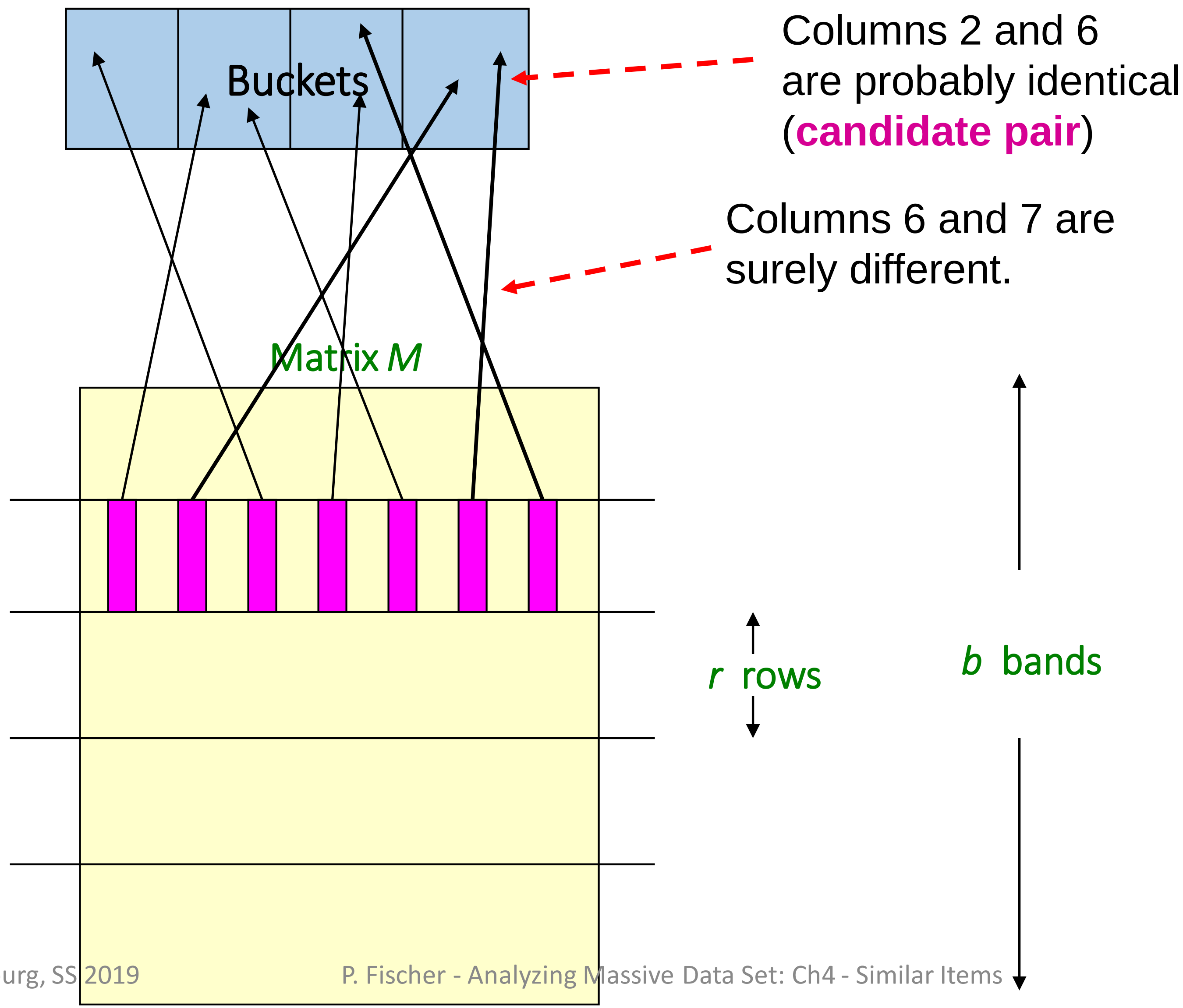
- Divide matrix $\boldsymbol{M}$ into $\boldsymbol{b}$ bands of $\boldsymbol{r}$ rows

- For each band, hash its portion of each column to a hash table with $\boldsymbol{k}$ buckets
  - Make $\boldsymbol{k}$ as large as possible

- ***Candidate*** column pairs are those that hash to the same bucket for $\geq$ **1** band

- Tune $\boldsymbol{b}$ and $\boldsymbol{r}$ to catch most similar pairs, but few non-similar pairs

Hashing Bands



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 ***not*** similar 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**

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

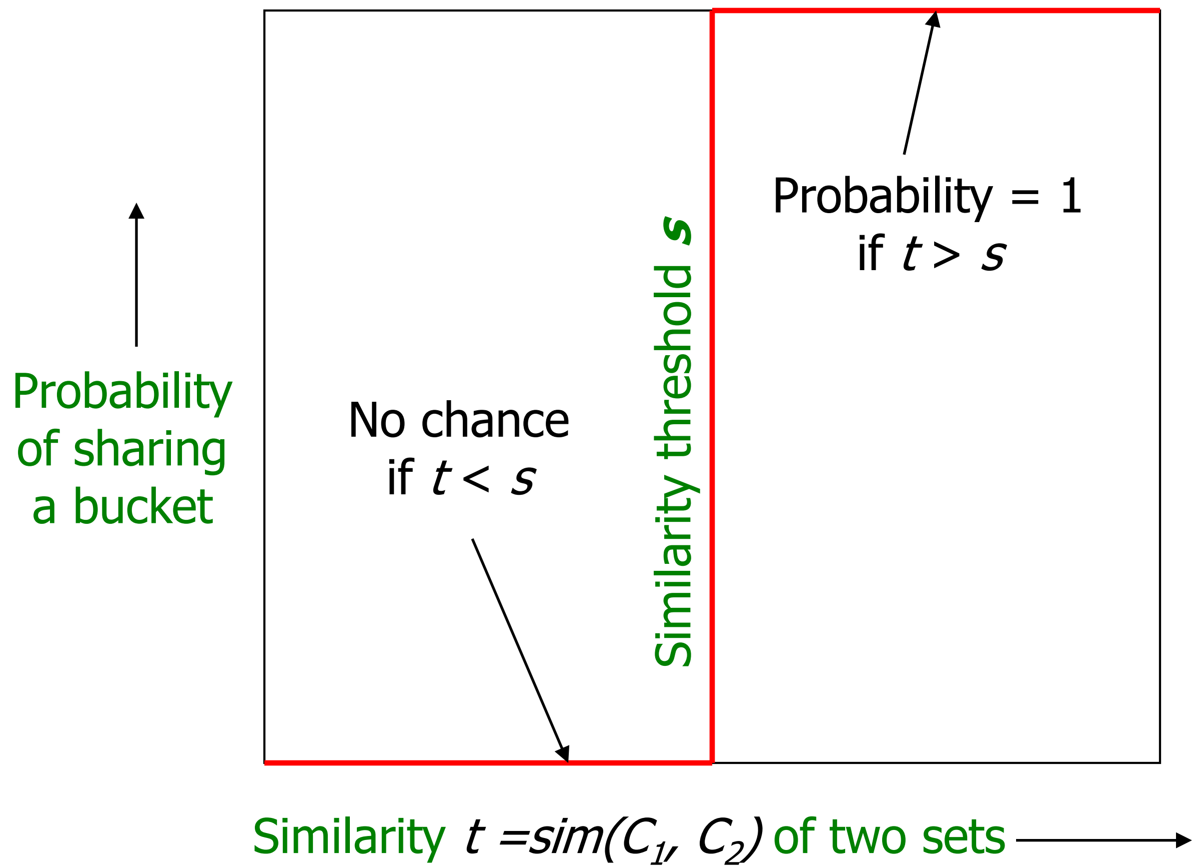
LSH Involves a Tradeoff

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

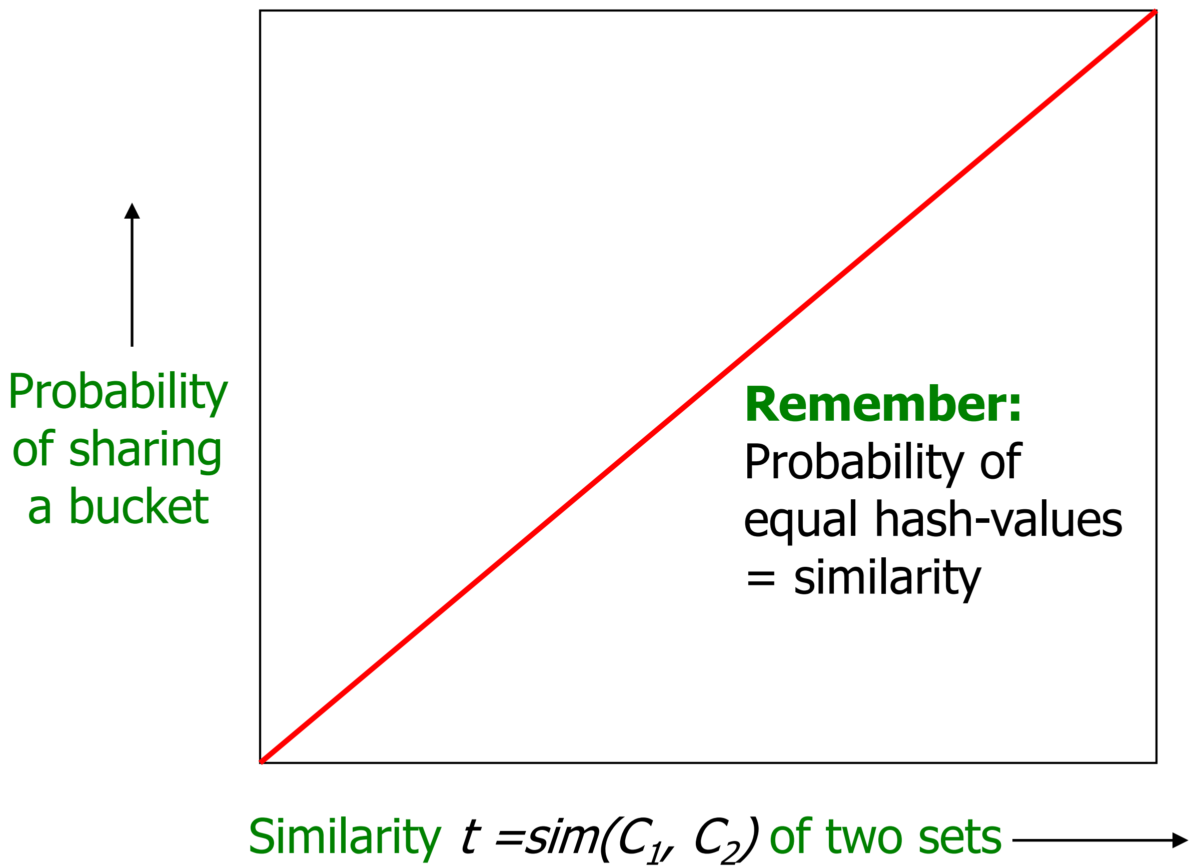
  to balance false positives/negatives

- **Example:** If we had only 15 bands of 5 rows, the number of false positives would go down, but the number of false negatives would go up

Analysis of LSH – What We Want



What 1 Band of 1 Row Gives You

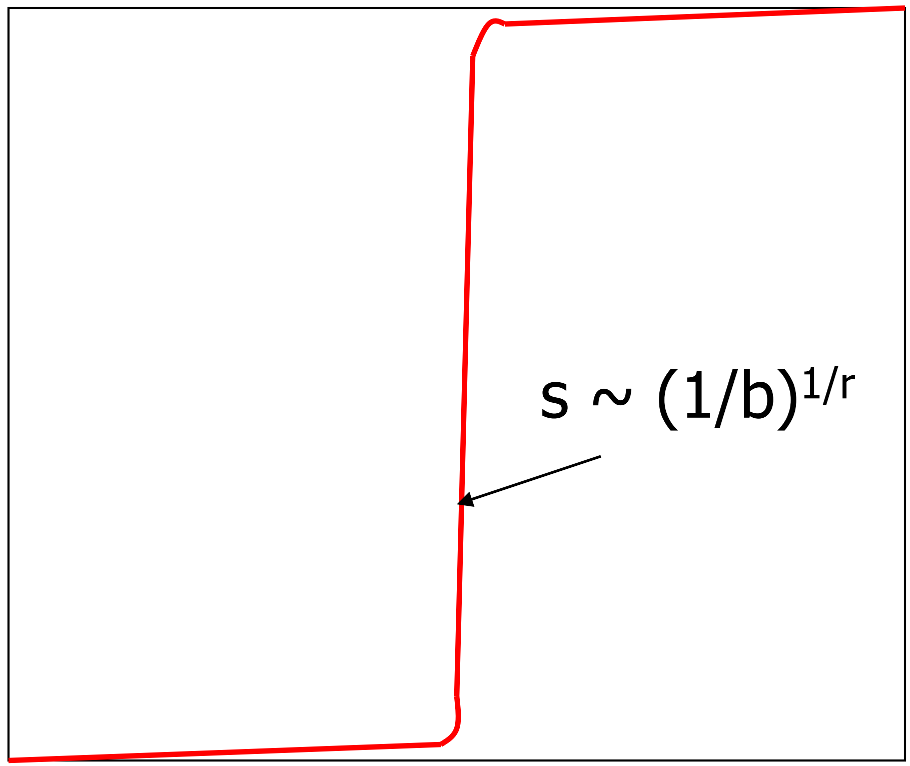


b bands, r rows/band

- Columns C_1 and C_2 have similarity $\boldsymbol{t}$
- Pick any band ($\boldsymbol{r}$ rows)
  - Prob. that all rows in band equal = $\boldsymbol{t^r}$
  - Prob. that some row in band unequal = $\mathbf{1 - }\boldsymbol{t^r}$
- Prob. that no band identical = $\mathbf{(1 - }\boldsymbol{t^r}\mathbf{)}^{\boldsymbol{b}}$
- Prob. that at least 1 band identical = $\mathbf{1 - (1 - }\boldsymbol{t^r}\mathbf{)}^{\boldsymbol{b}}$

What b Bands of r Rows Gives You

Probability of sharing a bucket



Similarity $t=sim(C_1, C_2)$ of two sets

$$\mathbf{1 - (1 - \boldsymbol{t}^{\boldsymbol{r}})^{\boldsymbol{b}}}$$

At least one band identical

No bands identical

Some row of a band unequal

All rows of a band are equal

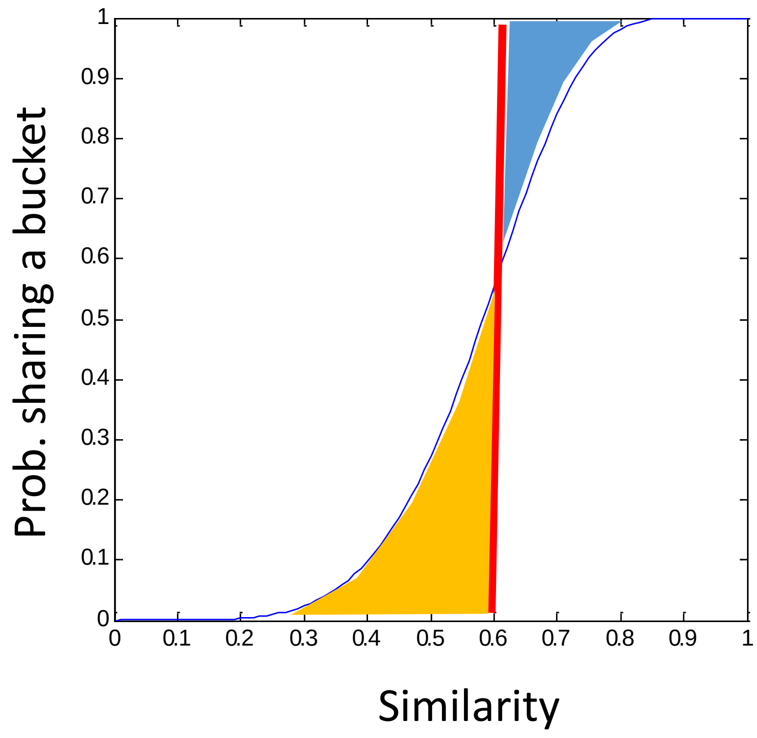
Example: $b = 20$; $r = 5$

- **Similarity threshold s**
- **Prob. that at least 1 band is identical:**

s	$1-(1-s^r)^b$
.2	.006
.3	.047
.4	.186
.5	.470
.6	.802
.7	.975
.8	.9996

Picking *r* and *b*: The S-curve

- **Picking *r* and *b* to get the best S-curve**
  - 50 hash-functions (r=5, b=10)



Blue area: False Negative rate
Yellow area: False Positive rate

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