Finding Similar Items

Overview

- Introduction
- Shingling
- MinHashing
- Locality Sensitive Hashing (LSH)

Introduction

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 (plagiarism, mirrors), classification by topic
  - **Customers who purchased similar products**
    - Products with similar customer sets (Reccommendation)
  - **Images with similar features**
    - Users who visited similar websites

A frequent issue

Given O objects, described with a set of **d** features, the goal is to find groups of *similar objects*

Objects = Users, pages, tweets, products, trajectories,...

Features = measurable characteristics, such as binary, categorical or real

Similarity(o1,o2) is a function that, taken the set of features of users u1 and u2, returns a value in [0,1]

Operations should be supported fast => without scanning the whole dataset (query), or take almost linear time (clustering)

Example #1: Users

Here O = U users

Features = Personal data, preferences (restaurants, books, products,...), navigational/search behavior,...

	Brahma Bull	Spaghetti House	Mango	Il Fornaio	Zao	Ming's	Ramona's	Straits	Homma's
Alice		Yes	No	Yes				No	
Bob		Yes				No		No	
Cindy				Yes	No			No	
Dave	No			No	Yes	Yes			Yes
Estie				No	Yes	Yes		Yes	
Fred	No						No		

Find users similar to Dave

Cluster similar users

Similarity = Hamming distance

Hamming distance

- Hamming distance measures the number of positions where two equal-length sequences differ.
- Example (Binary):

A = 1011101

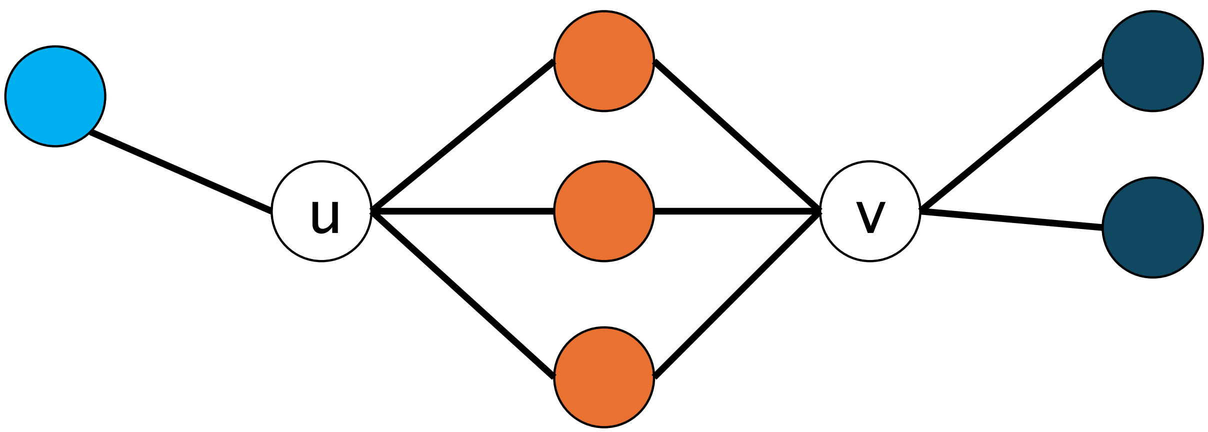
B = 1001001

Hamming distance = 2

Example #2: Graph

Here O = G(V,E) graph

Features = adjacency lists, very sparse



Find nodes similar to u

Cluster similar nodes

Jaccard coeff $\frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|}$

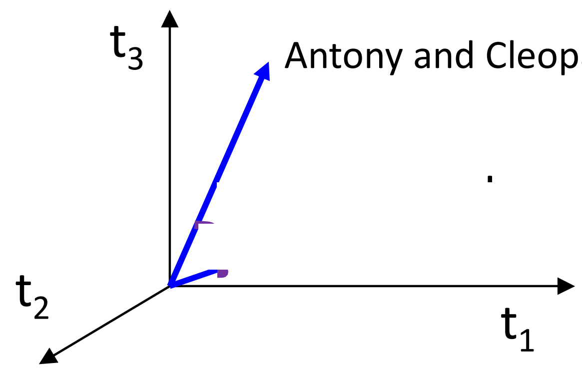
Example #3: Documents

Here O = documents

Features = tf-idf of words, **d is very very large**

	Antony and Cleopatra
Antony	13,1
Brutus	3,0
Caesar	2,3
Calpurnia	0,0
Cleopatra	17,7
mercy	0,5
worser	1,2

Vector Space model



Find documents similar to query

Cluster similar documents

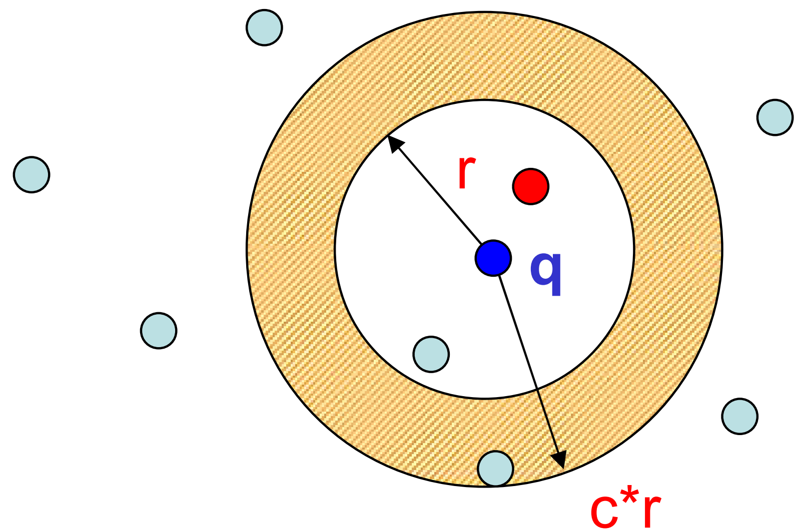
Cosine similarity =

dot product between two document vectors

Example #4: geometry

Here O = d-dimensional points

Features = coordinates, **possibly real**

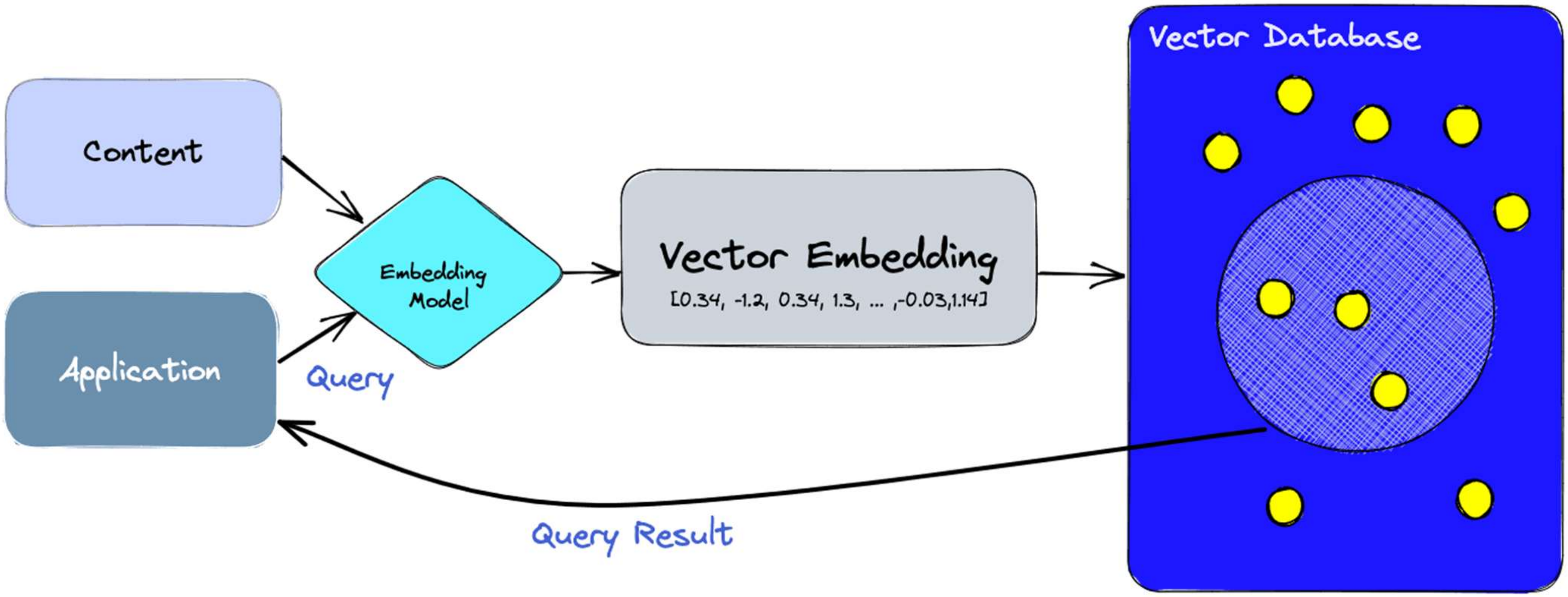


Find (r,c)-Near Neighbor to query **q** according to Euclidean

Find (FNE) Nearest Neighbor to query **q** according to Euclidean

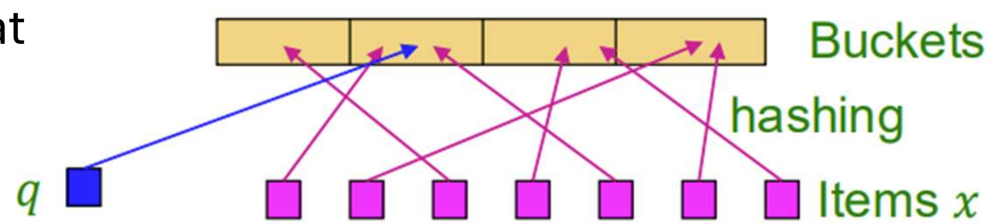
distance or other norms

Vector space embedding and LSH



https://www.pinecone.io/learn/vector-database/

- LSH is really a family of related techniques
- In general, one throws items into buckets using several different "hash functions".
- You examine only those pairs of items that share a bucket for at least one of these hashings.



Our problem

- **Given: High dimensional data points $\boldsymbol{x_1, x_2, \ldots}$**
  - **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 $\boldsymbol{d(x_1, x_2)}$**
  - Which quantifies the "distance" between $\boldsymbol{x_1}$ and $\boldsymbol{x_2}$

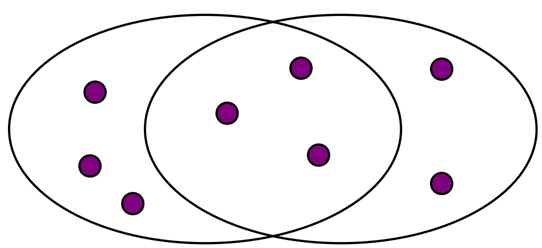
- **Goal:** Find **all pairs of data points** $(\boldsymbol{x_i, x_j})$ that are within some distance threshold $\boldsymbol{d(x_i, x_j) \leq s}$
- **Note:** Naïve solution would take $\boldsymbol{O(N^2)}$ ☹

  where $\boldsymbol{N}$ is the number of data points

- **MAGIC: This can be done in $\boldsymbol{O(N)}$!! 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
- **Example: 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:** $d(C_1, C_2) = 1 - |C_1 \cap C_2| / |C_1 \cup C_2|$



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

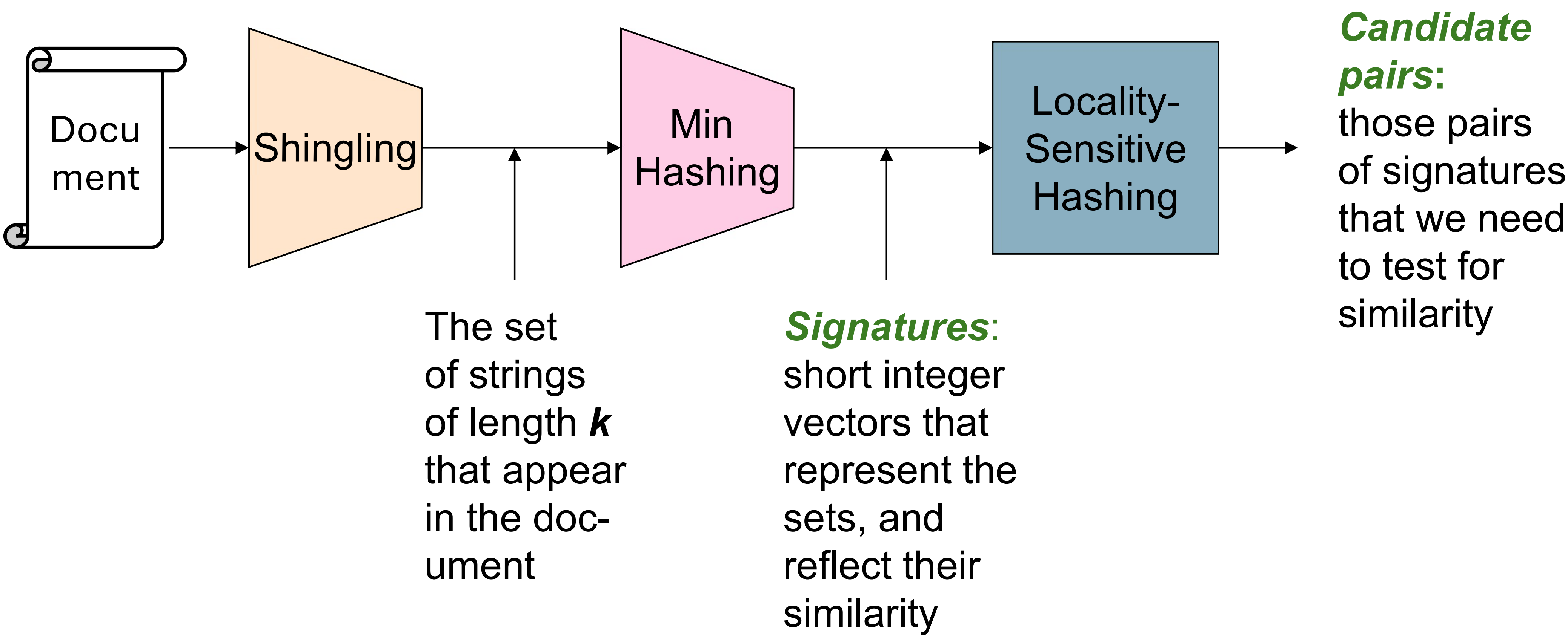
Task: Finding Similar Documents

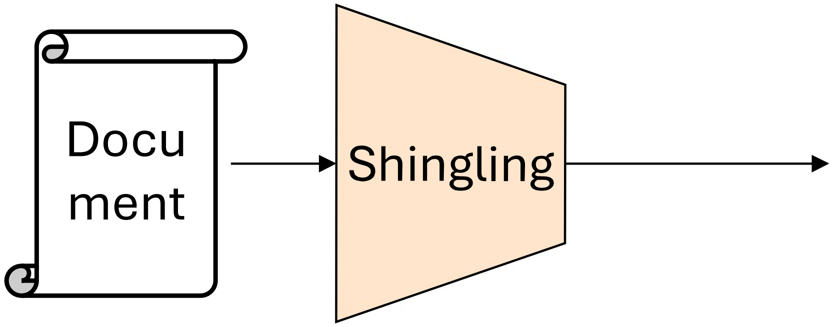
- **Goal: Given a large number (N in the millions or billions) of documents, find "near duplicate" pairs**
- **Applications:**
  - Plagiarism
  - Mirror websites, or approximate mirrors (Don't want to show both in search results)
  - Similar news articles at many news sites (Cluster articles by "same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

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) 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}

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

How large K should be?

- It depends on how long typical documents are and how large the set of typical character is.
  - Idea: k should be picked large enough that the probability of any given shingle appearing in any given document is low.
- **Example:**
  - ❖e-mails: only letters and white-space character
  - ➢With **k**=5 there would be $27^5 = 14M$ shingles
    - **Note:** an email usually is much smaller than 14M chars.

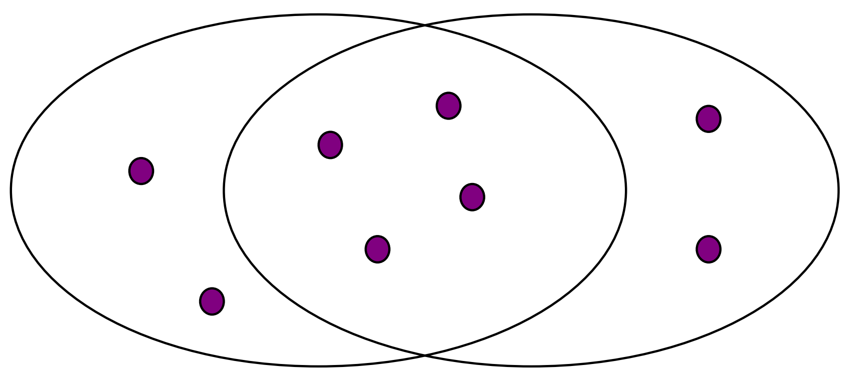
Compressing Shingles

- 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**
- **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

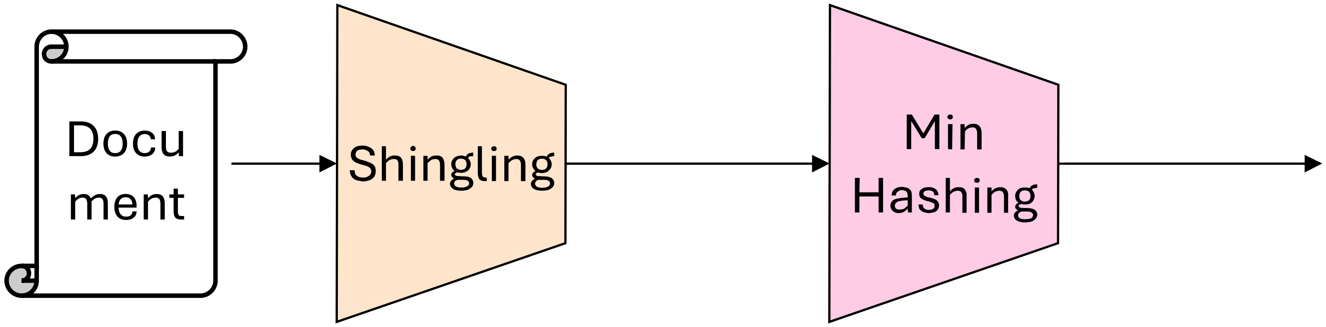
- **Document D_1 is a set of its k-shingles $C_1=S(D_1)$**
  - Equivalently, each document is a 0/1 vector in the space of *k*-shingles
    - Each unique shingle is a dimension
    - Vectors are very sparse
- **A natural similarity measure is the Jaccard similarity:**

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



Motivation for Minhash/LSH

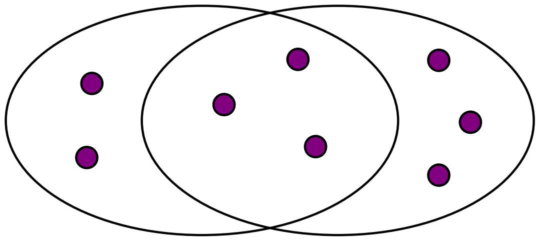
- **Suppose we need to find near-duplicate documents among $N = 1$ million documents**
- Naïvely, we would have to compute **pairwise Jaccard similarities** for **every pair of docs**
  - $\boldsymbol{N(N-1)/2} \approx$ **5*10^{11}** comparisons
  - At 10^5 secs/day and 10^6 comparisons/sec, it would take **5 days**
- For $\boldsymbol{N = 10}$ million, it takes more than a year…



Convert large sets to short signatures, while preserving similarity

Min-Hashing

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)
  - $M[e, s] = 1$ 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 $h(\cdot)$ 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

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

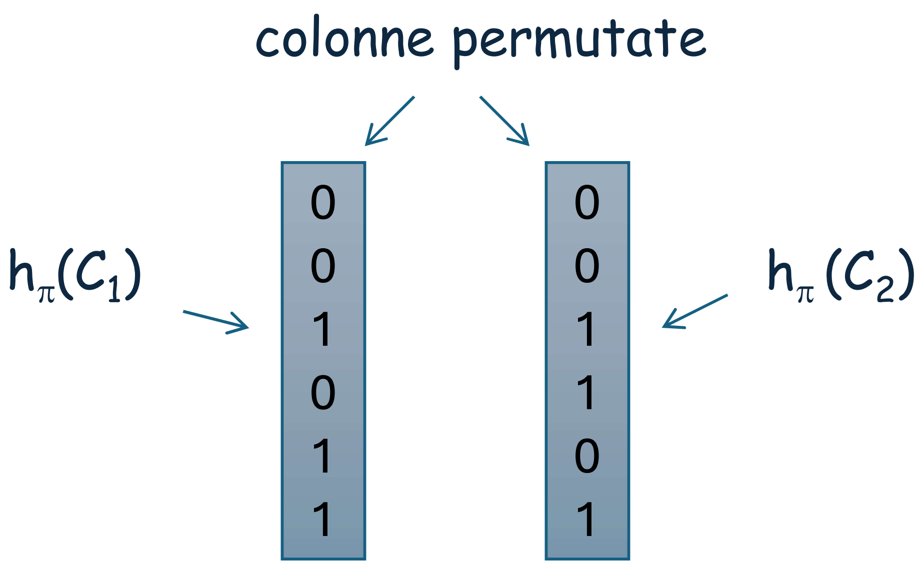
- **Choose a random permutation π**
- **<u>Claim:</u> $\Pr[h_\pi(C_1) = h_\pi(C_2)] = sim(C_1, C_2)$**
- **Why?**
  - Let **X** be a doc (set of shingles), $\boldsymbol{y} \in \boldsymbol{X}$ is a shingle
  - **Then: $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$**
    - It is equally likely that any $\boldsymbol{y}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

Analisi

- Qual è la probabilità che $h_l(C_1) = h_l(C_2)$?



$$\Pr ob[h_l(C_1) = h_l(C_2)] = \frac{\text{righe con entrambe le colonne 1}}{\text{righe con almeno una delle due colonne 1}} = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} = Sim(C_1, C_2)$$

Analisi

Qual è la similarità tra $Sig(C_1)$ e $Sig(C_2)$?

$Sig(C_1)$	$Sig(C_2)$
$h_1(C_1)$	$h_1(C_2)$
$h_2(C_1)$	$h_2(C_2)$
…	…
…	…
$h_k(C_1)$	$h_k(C_2)$

$$Sim(Sign(C_1), Sign(C_2)) = \frac{\text{numero di righe in cui } h_l(C_1) = h_l(C_2)}{\text{numero totale di righe}}$$

$$Sim(Sign(C_1), Sign(C_2)) \longrightarrow \Pr ob[h_l(C_1) = h_l(C_2)] = Sim(C_1, C_2)$$

Correttezza del Min-Hashing

- Definiamo $Sim^*(C_i,C_j) = \mathrm{Sim}(\mathrm{Sig}(C_i),\mathrm{Sig}(C_j))$
- Sia *s* una soglia di similarità al di sopra della quale consideriamo due colonne "altamente simili". Supponiamo che *s* sia limitato inferiormente da una costante *c*.

- **Teorema:** Siano $0 < \delta < 1$, $\varepsilon > 0$, $k > 2\,\delta^{-2} c^{-1} \log \varepsilon^{-1}$. Allora per ogni coppia di colonne C_i , C_j valgono le seguenti proprietà:
  a. Se $Sim(C_i,C_j) \geq s \geq c$, allora $\mathrm{Prob}[Sim^*(C_i,C_j) \geq (1-\delta)\, s] \geq 1 - \varepsilon$
  b. Se $Sim(C_i,C_j) \leq c$ allora $\mathrm{Prob}[Sim^*(C_i,C_j) \leq (1+\delta)\, c] \geq 1 - \varepsilon$

- Il teorema ci dice che il numero di falsi positivi e negativi è limitato

Dimostrazione

- Dimostriamo la proprietà *a*. Per la *b* la dimostrazione è simile.
- Fissata una coppia di colonne C_i, C_j con $Sim(C_i,C_j) \geq s$ dobbiamo dimostrare che:

$$\text{Prob}[Sim^*(C_i, C_j) < (1-\delta)\cdot s] < \varepsilon$$

- Consideriamo la variabile aleatoria $X = X_1+\ldots+X_k$ dove X_l vale 1 se $h_l(C_i) = h_l(C_j)$ altrimenti vale 0.
- Il Chernoff Bound ci dice che:

$$\text{Prob}[X < (1-\delta)\cdot E[X]] < e^{-\frac{\delta^2 E[X]}{2}}$$

Dimostrazione

- Inoltre:
  - $Sim^*(C_i,C_j) = X / k$
  - $E[X] = k * Sim(C_i,C_j)$ (previsione di X)
- Quindi:

$$\mathrm{Prob}[Sim^*(C_i,C_j) < (1-\delta)\cdot s] = \mathrm{Prob}[\frac{X}{k} < (1-\delta)\cdot s] =$$

$$= \mathrm{Prob}[X < (1-\delta)\cdot k \cdot s] \le \mathrm{Prob}[X < (1-\delta)\cdot E[X]] <$$

$$< e^{-\frac{\delta^2 \cdot E[X]}{2}} = e^{-\frac{\delta^2 \cdot k \cdot Sim(C_i,C_j)}{2}} \le e^{-\frac{\delta^2 \cdot k \cdot s}{2}} < \varepsilon$$

Similarity for 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**

- **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	3-4
Col/Col	0.75	0.75	0	0
Sig/Sig	0.67	1.00	0	0

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*

  $$\boldsymbol{sig(C)[i] = \min(\pi_i(C))}$$

- **Note:** The sketch (signature) of document *C* is small **$\sim$100 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 k_i
  - Ordering under k_i gives a random row permutation!
- **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 $sig(C)[i] \leftarrow k_i(j)$

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)

Row	C_1	C_2	C_3	C_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

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

	C_1	C_2	C_3	C_4
h_1	1	3	0	1
H_2	0	2	0	0