17 - Finding Similar Items (2)

Handling Large and Huge sets

Handling Large and Huge sets

- **Goal: Given a large number of documents (N), find "near duplicate" pairs**
  - N = **millions or billions**

- **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"

Finding Similar Documents

- **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

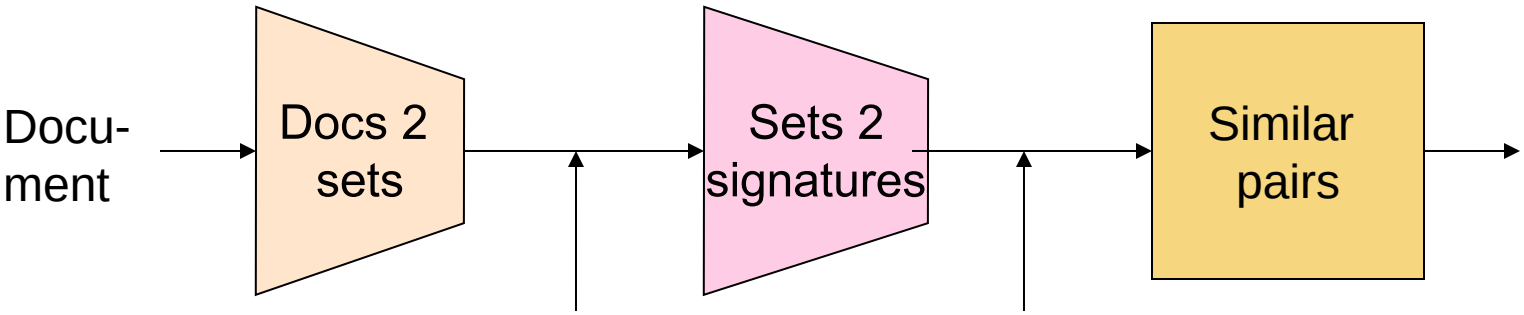
3 Essential Steps for Similar Docs

1. Convert documents **to representative sets**

2. Convert sets **to short signatures**,
   while preserving similarity

3. Determine pairs of signatures likely to be from similar documents
   - **Candidate pairs!**

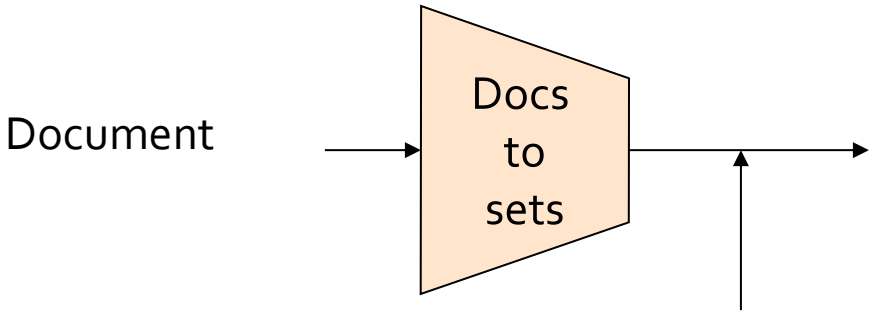
The Big Picture



The set of strings of length ***k*** that appear in the doc-ument

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

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



Representative set

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

Step 1: Convert documents to sets → ***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.
- **Need to account for ordering of words!**
- A different way: **Shingles!**

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 **D_1** = abcab

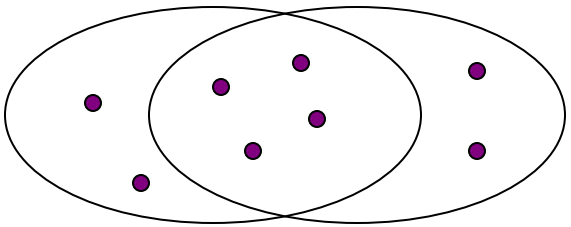
- Set of 2-shingles: **$S(D_1)$** = {ab, bc, ca}

- **Other possibility:**
- Shingles as a bag (multiset),
  - count ab twice: **$S'(D_1)$ =** {ab, bc, ca, ab}

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:**

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

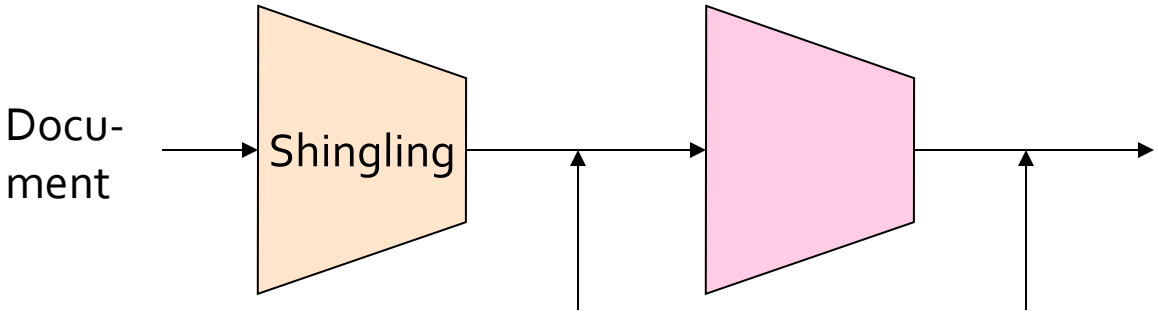


Working Assumption

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

Problem not solved yet ☹

- **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**
  - $\boldsymbol{N(N-1)/2}$ **≈ 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…
- We need badly to make this much faster…



Signatures:

short representation of the sets, that reflect their similarity

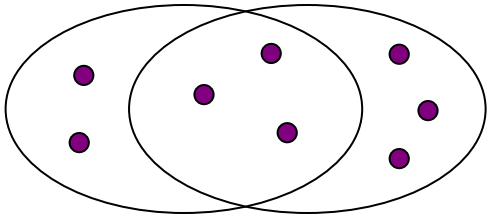
---

Step 2: Convert **large sets** to **short signatures**,

while **preserving similarity**

Encoding Sets as Bit Vectors

- As seen before, many similarity problems can be formalized as **finding subsets that have significant intersection**



- **It is important to determine intersection fast**

- **Idea: Encode sets using 0/1 (bit, boolean) vectors**
  - One dimension per element in the universal set

- set intersection is bitwise **AND**
- set union can be obtained by bitwise **OR**

Example

- C_1 = 10111; C_2 = 10011
- Size of intersection?
- **3**
- Size of union?
- **4**
- **Jaccard similarity** (not distance) **= ¾**
- **Distance:**
  - **$d(C_1,C_2)$ = 1 – (Jaccard similarity) = 1/4**

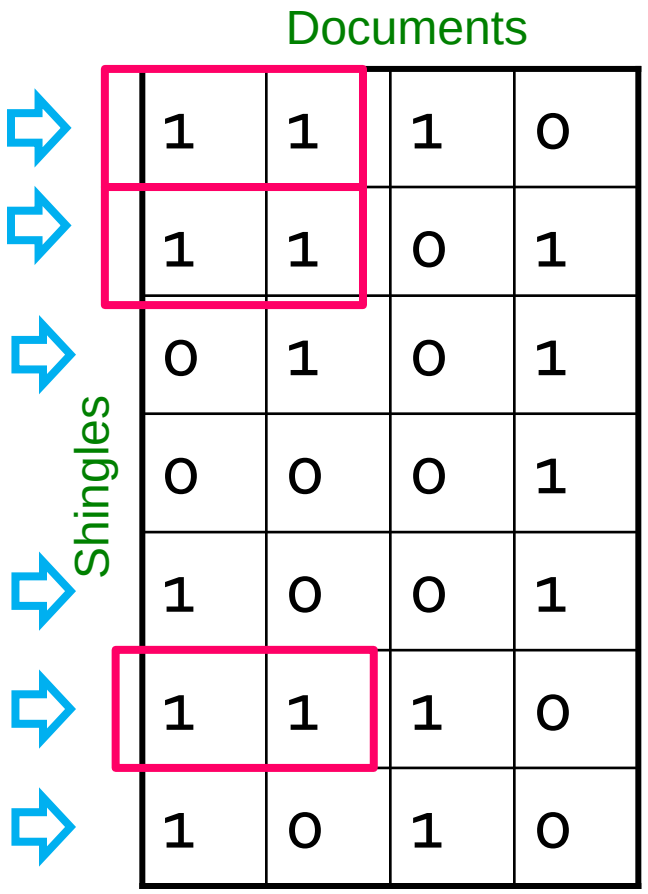
From Sets to Boolean Matrices

- **Rows** = elements (ex: 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!**

Shingles \ Documents				
	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

Example

- **Each document is a column**
  - C_i is the ith Document
- **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**



Next: Finding Similar Columns

- **So far:**
  - Documents → Sets of shingles
  - Represent sets as boolean vectors in a matrix

- **Next goal: Find similar columns while computing small signatures**
  - **Similarity of columns == similarity of signatures**

Signatures

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)***

Hashing Columns (Signatures)

- **Find a hash function *h(·)* such that:**
  - If ***sim*(C_1,C_2)** is high, then with high prob. ***h*(C_1) = *h*(C_2)**
  - If ***sim*(C_1,C_2)** is low, then with high prob. ***h*(C_1) ≠ *h*(C_2)**

- **Hash docs into buckets. Expect that "most" pairs of near duplicate docs hash into the same bucket!**

Min-Hashing

- **Goal: Find a hash function *h(·)* 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 row** (in the permuted order π) in which column C has value **1**:

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

- Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column

Min-Hashing Example

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

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

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

Permutation π | Input matrix (Shingles x Documents) | Signature matrix *M*

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

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

The Min-Hash Property

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

- **Why?**

- Let **X** be a doc (set of shingles), $y \in X$ is a shingle

- It is equally likely that any $y \in X$ is mapped to the ***min*** element

- **Then:** $\Pr[\pi(y) = \min(\pi(X))] = 1/|X|$

The Min-Hash Property

- Size of the universe of all possible values of $\min(\pi(C_1 \cup C_2))$ is $|C_1 \cup C_2|$
  - All shingles in $C_1 \cup C_2$ can be the minimum

- In $|C_1 \cap C_2|$ of the cases we can have $\min(\pi(C_1))=\min(\pi(C_2))$

- In consequence:
- **$\Pr[\min(\pi(C_1))=\min(\pi(C_2))]=|C_1 \cap C_2|/|C_1 \cup C_2|$**
  - **The Jaccard similarity**

The Min-Hash Property

- For two columns A and B:

A	B
0	0
0	0
1	**1**
0	0
0	1
1	0

- h(A) = h(B) exactly when the minimum hash value of the union A ∪ B lies in the intersection A ∩ B.

demoMinhHashProperty.m

Similarity of Signatures

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

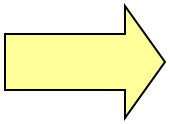
Min-Hashing Example

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

Min-Hash Signatures

- **Pick K random permutations of the rows**
- Think of signature ***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 **signature of document** *C* is small
  - **~100 bytes for K=100** ☺

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

Making it practical

Implementation

- **Permuting rows even once is prohibitive**
- **Use hashing instead!**
  - Pick **K** hash functions h_i
    - *Ex:* **K = 100**
- Ordering under h_i gives a random row permutation!

Implementation

- **Good News:**
  - We can use directly the miminum of the hash codes of the elements of our sets (S)

- Let *h* be a hash function that maps the members of C_1 and C_2 to distinct integers
- *Let* $h_{\min}(S)$ be the minimal member of *S* with respect to *h*
  - that is, the member *x* of *S* with the minimum value of *h*(*x*)

Implementation

- If we apply h_{min} to both C_1 and C_2, we will get the same value exactly when the element of the union ($C_1 \cup C_2$) with minimum hash value lies in the intersection ($C_1 \cap C_2$)

- The probability of this being true is:

$$|C_1 \cap C_2| / |C_1 \cup C_2|$$

- and therefore:

Pr[$h_{\text{min}}(C_1) = h_{\text{min}}(C_2)$] = *Jaccard Similarity*(C_1, C_2)

Implementation:Several hash functions

- That is, the probability that $h_{\min}(C_1) = h_{\min}(C_2)$ is true is equal to the Jaccard similarity
- Defining the random variable *X* as one when $h_{\min}(C_1) = h_{\min}(C_2)$ and zero otherwise, then *X* is an (unbiased) estimator of *J*accard Similarity
- By being only zero or one, *X* has too high a variance to be a useful estimator
- The idea of **the MinHash scheme** is to **reduce this variance by averaging together several variables** constructed in the same way
  - Estimate is equal to fraction of hash functions that agree

Implementation: Random hash functions

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

Demonstration

- Not convinced ?
- Let’s try
  - Similarity estimation using minimum of the hash codes
  - Movielens: Replacing Jaccard Distance computation by an estimate of the distance based on minhash…

To be continued ...

Part of the slides Adapted from:
Finding Similar Items: Locality Sensitive Hashing

Mining of Massive Datasets
Jure Leskovec, Anand Rajaraman, Jeff Ullman
Stanford University
http://www.mmds.org