Finding Near-Duplicates

Mining Massive Datasets

Prof. Carlos Castillo — https://chato.cl/teach

Universitat Pompeu Fabra Barcelona

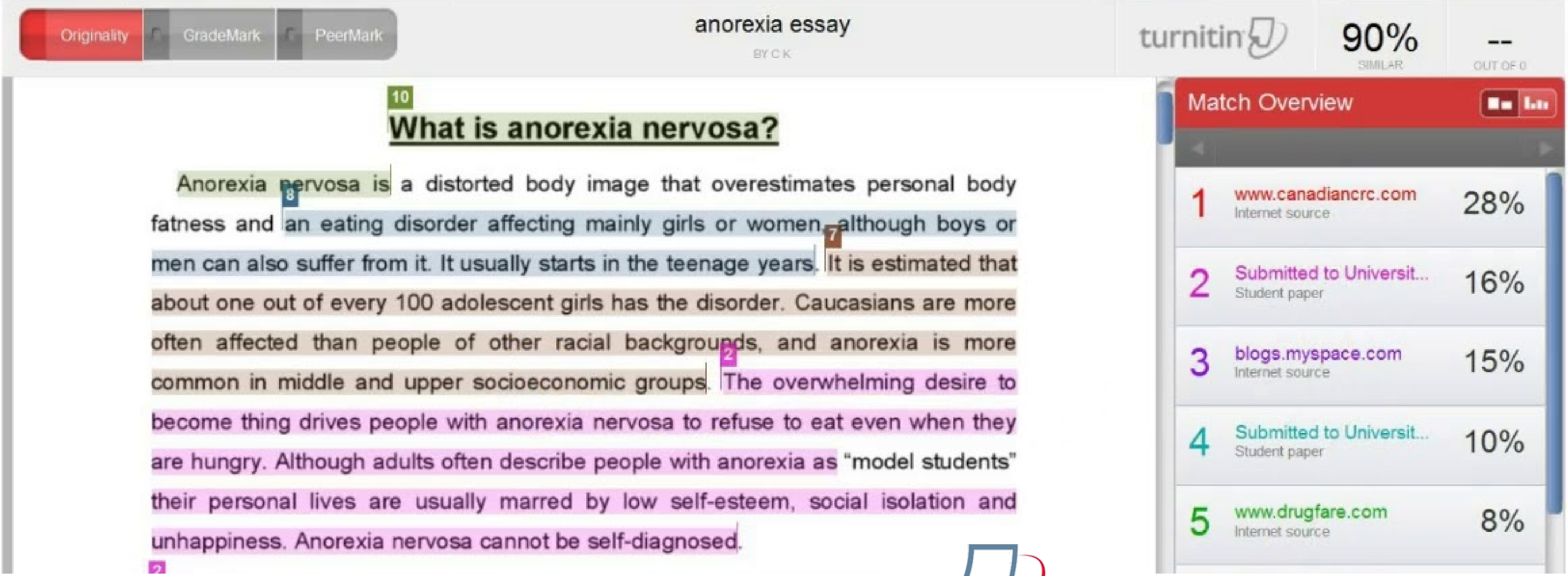
Source for this deck

- Mining of Massive Datasets 2nd edition (2014) by Leskovec et al. (Chapter 3) [slides ch3]

Fast near-neighbor applications

- For documents
  - Find “legitimate” duplicates
    - Copies of the same press release or cable
    - Mirrors of the same documents, for efficiency
  - Find “illegitimate” duplicates
    - Plagiarism
- For baskets
  - Find customers who purchase similar items

Example: plagiarism detection

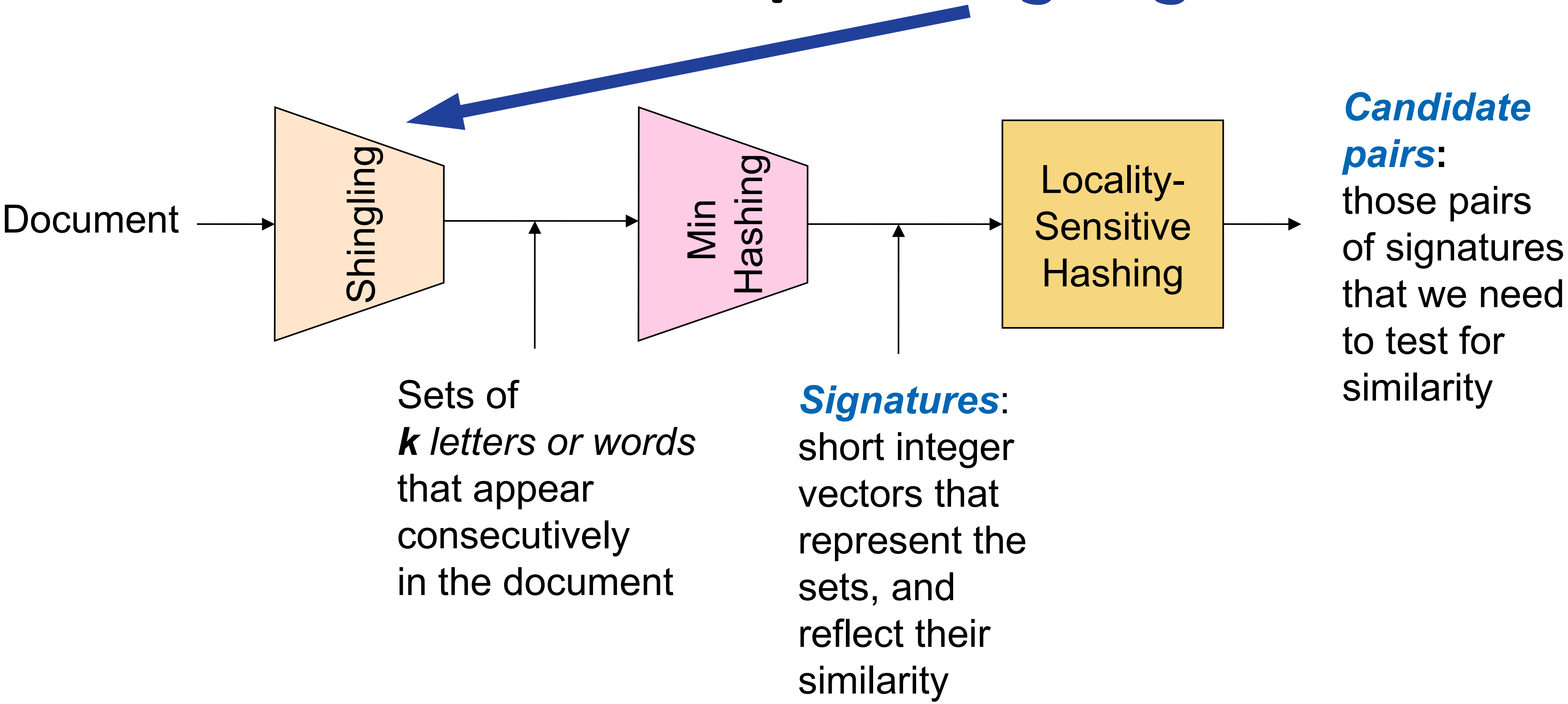


Fast near-neighbor challenges

- Too many documents to compare all pairs
  - OK to pay linear or log cost, but not quadratic
- Documents cannot fit in main memory
  - They are too large or too many
- Many small pieces of one document can appear out of order in another

Shingling
(ngrams)

First step: shingling



Naïve solution:
feature selection over bag of words

- Document = set of terms

  $\rightarrow$ Document = set of important terms

- Now, compute all pairs similarity

- Doesn't work for at least two reasons, why?

Naïve solution: feature selection over bag of words

- Document = set of terms

  $\rightarrow$ Document = set of important terms

- Now, compute all pairs similarity
- Doesn't work for at least two reasons, why?
  - Doesn't preserve the ordering
  - Unimportant terms are also relevant (stylistic)

Shingles

- An **n-gram** in a document is a sequence of *n* tokens that appears in the document
- **Shingles** are either n-grams (word-level) or sequences of characters (“character n-grams”), depending on the application
- Character-level example: k=2; document D1 = abcab
  Set of 2-shingles: S(D1) = {ab, bc, ca}
  - Option: Shingles as a bag (multiset), count ab twice: S’(D1) = {ab, bc, ca, ab}

Example: 4-grams

E.g., 4-shingles of

“`My name is Inigo Montoya. You killed my father. Prepare to die`”:

{

- `my name is inigo`
- `name is inigo montoya`
- `is inigo montoya you`
- `inigo montoya you killed`
- `montoya you killed my`
- `you killed my father`
- `killed my father prepare`
- `my father prepare to`
- `father prepare to die`

}

Compressed representation of shingles

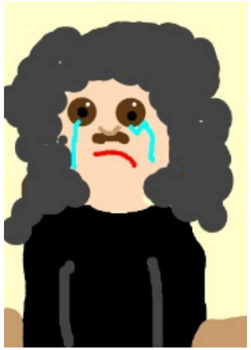
- To compress long shingles, we can **hash** them to (say) 4 bytes
- **Represent a document by the set of hash values of its k-shingles**
  - Note we could have false positives due to hash collisions
- Example: k=2; document D1= abcab
  Set of 2-shingles: S(D1) = {ab, bc, ca}
  Hash the singles (example): h(D1) = {1, 5, 7}

Documents as sets of shingles

- A document is now **a set of shingles**
  - Dimensionality reduced from "words in a dictionary" to "number of distinct shingles"
  - Higher dimensionality but more sparse
- Working assumption
  - **Documents that have lots of shingles in common have similar text**, even if the text appears in different order
- In practice, k should be large enough, or most documents will have most shingles
  - $k = 5$ is OK for short documents
  - $k = 10$ is better for long documents

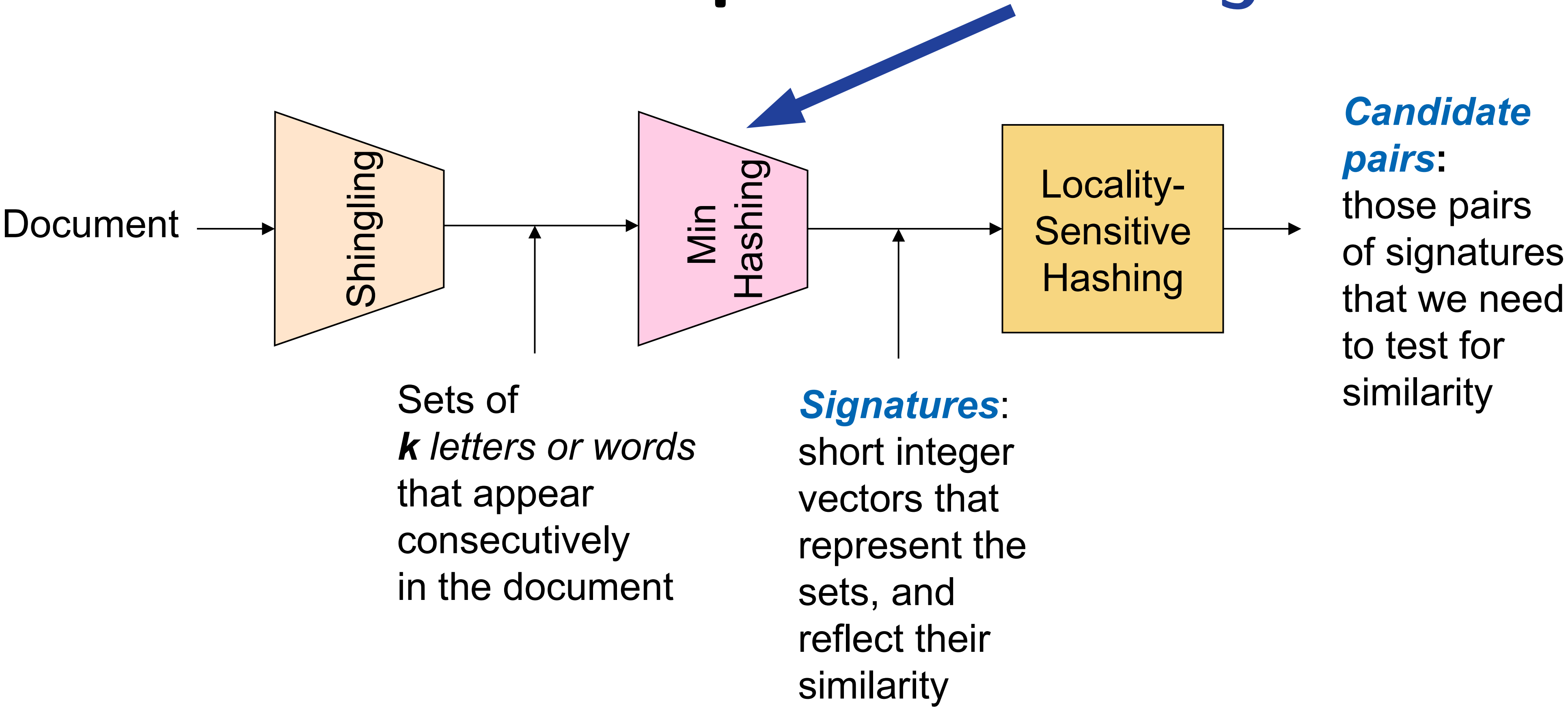
Using shingles directly

- Suppose we need to find near-duplicate documents among one million documents
- Naïvely, we would have to compute all pairwise Jaccard similarities $\approx 5*10^{11}$ comparisons
- At 10^5 secs/day and 10^6 comparisons/sec, it would take 5 days
- For 10 million documents, it takes more than a year…



Min hashing

Next step: min hashing



Sets can be bit vectors

- Many similarity problems involve **finding subsets with substantial intersection**

- Remember we can encode sets using bit vectors
  - set intersection = bitwise **AND**
  - set union = bitwise **OR**

- **Example:** $C_1 = 10111$; $C_2 = 10011$
  - 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**

$$J(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$

From sets to boolean matrices

- Rows = items (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 very sparse!

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

Hashing set representations

- We don't want to compare c_1, c_2, they might be too large, slowing down the computation
- Instead, we compute signatures $h(c_1)$, $h(c_2)$ that are smaller in size than c_1 and c_2
- **Desired properties:**
  $c_1 = c_2 \Rightarrow \text{Prob.}(h(c_1) = h(c_2))$ is large
  $c_1 \neq c_2 \Rightarrow \text{Prob.}(h(c_1) \neq h(c_2))$ is large

Hashing set representations (cont.)

- Naïve approach (non-LSH-based):
  - 1) Compute signatures of columns: small summaries of columns
  - 2) Examine all pairs of signatures to find similar columns
    - Essential: Similarities of signatures and columns are related
  - 3) Optional: verify 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)

Hash function for Jaccard metric: min hashing

- Imagine the rows of the boolean matrix permuted under **random but fixed 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**
- Use several (e.g., 100) independent hash functions (that is, permutations) to create a signature of a column

Minhash example

Permutations

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

Rows=Shingles, Columns=Documents

D1	D2	D3	D4
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*

D1	D2	D3	D4
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

Exercise: shingling

Rows=Shingles, Columns=Documents

Permutation	D1	D2	D3	D4
3	1	0	1	0
2	1	0	0	1
1	0	1	0	1
4	0	1	0	1
7	0	1	0	1
5	1	0	1	0
6	1	0	1	0

Signature matrix M

D1	D2	D3	D4

Index of the bit vector position where the first 1 occurs according to the ordering of the permutation

Pin board: https://upfbarcelona.padlet.org/chato/bu8ekcrferwf6lv5

Minhash approximates Jaccard

- Let π be a random permutation
- Let $h_\pi(S)$ be the first element of S under the permutation π
- If $h_\pi(A) = h_\pi(B)$ and there are no collisions, then:
  - Among all elements in A ∪ B …
  - … the chosen element is in A ∩ B
- This happens with probability $|A \cap B|/|A \cup B| = \text{Jaccard}(A, B)$
- Hence $\mathbf{Pr[h_\pi(A) = h_\pi(B)] = Jaccard(A, B)}$

We will use multiple permutations

- Jaccard(A, B) $= \mathrm{E}[h_\pi(\mathrm{A}) = h_\pi(\mathrm{B})]$

  $=$ number of matches / number of permutations

- We will use many permutations, e.g., 100

Example: three permutations

Permutations

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

Rows=Shingles, Columns=Documents

D1	D2	D3	D4
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

D1	D2	D3	D4
2	1	2	1
2	1	4	1
1	2	1	2

Similarities	1-3	2-4	1-2	3-4
Complete	0.75	0.75	0	0
Signatures	0.67	1.00	0	0

Minhash signatures

- **Pick $\pi_1 \ldots \pi_{100}$ random permutations of the rows (K=100)**
- Think of ***sig*(C)** as a column vector
  - **s*ig*(C)[i] =** according to the *i*-th permutation, the index of the first row that has a 1 in column *C*
  - $\boldsymbol{sig}\mathbf{(C)[i] = min\ (\pi_i(C))}$
- The signature or "sketch" of document *C* has fixed size!
  - We achieved our goal: we "compressed" long bit vectors into short signatures

Implementation

- **Permuting rows even once is prohibitive**
- Instead, we create $\boldsymbol{\pi_1}$ **...** $\boldsymbol{\pi_{100}}$ by using $\mathbf{K = 100}$ hash actual functions H_i ... which map to integer numbers

  Ordering of $\{1,2,\ldots,n\}$ under H_i

  ... i.e., computing h(1), h(2), ..., h(n) and sorting

  ... is a random permutation!

Summary

Things to remember

- **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

Exercises for TT08-TT09

- Mining of Massive Datasets 2nd edition (2014) by Leskovec et al.
  - Exercises 3.1.4 (Jaccard similarity)
  - Exercises 3.2.5 (Shingling)
  - Exercises 3.3.6 (Min hashing)
  - Exercises 3.4.4 (Locality-sensitive hashing)