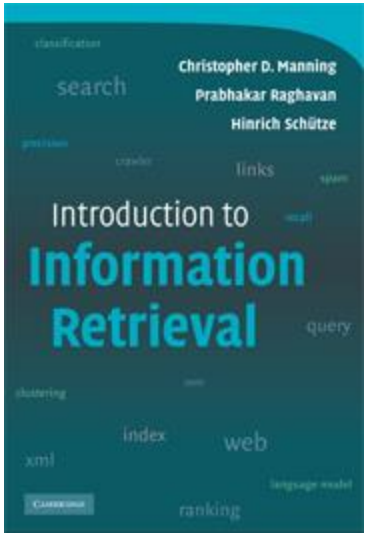
Information Retrieval and Organisation



Chapter 19.6

Near-Duplicates and Shingling

Dell Zhang
Birkbeck, University of London

Duplicate Documents

- The Web is full of duplicated content
- Exact duplicates (exact match)
  - Not so common
  - Easy to eliminate using hash/fingerprint etc.
- Near-duplicates (approximate match)
  - Many, many cases, e.g., last modified date the only difference between two copies of a page
  - Difficult to eliminate

Near-Duplicate Detection

- It is necessary to eliminate near-duplicates
  - For the user, it’s annoying to get a search result with near-identical documents
  - Marginal relevance is zero: even a highly relevant document becomes non-relevant if it appears below a (near-)duplicate
- How would you do that?

Near-Duplicate Detection

- Compute similarity between documents
  - We want "syntactic" (as opposed to semantic) similarity. That is to say, we do not consider documents near-duplicates if they have the same content but express it with different words.
- Detect near duplicates using a similarity threshold θ
  - For example, the documents with similarity $> \theta = 80\%$ are deemed to be near-duplicates
  - Not really transitive, though sometimes regarded as transitive for convenience

Feature Representation

- Represent each document as a set of **shingles** (word k-grams)

  "<u>*a rose is a rose is a rose*</u>" ➔ 4-grams

  a_rose_is_a

  rose_is_a_rose

  is_a_rose_is

  a_rose_is_a

  { *a_rose_is_a, rose_is_a_rose, is_a_rose_is* }

- Each distinct shingle s can be mapped to an m-bit *fingerprint* (e.g., m=64)
  - From now on, s refers to the shingle's fingerprint

Similarity Measure

- Define the syntactic similarity of two documents as the **Jaccard coefficient** of their shingle sets
  - = size_of_intersection / size_of_union
  - Note: very sensitive to syntactic dissimilarity

For example,

D_1: “Jack London travelled to Oakland”

D_2: “Jack London travelled to the city of Oakland”

D_3: “Jack travelled from Oakland to London”

Based on shingles of size 2 (2-grams or bigrams),

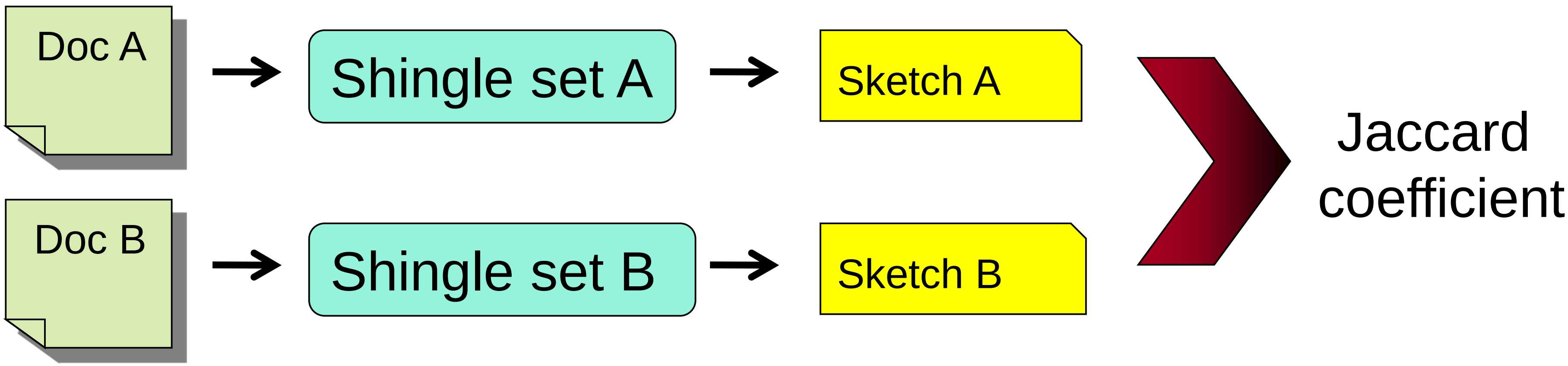
$J(D_1, D_2) = 3/8 = 0.375$

$J(D_1, D_3) = 0$

Computing Similarity

- The number of shingles per document is large
- Computing the <u>exact</u> set intersection of shingles between a pair of documents is expensive
- So we approximate using a **sketch** --- a cleverly chosen *subset* of shingles from a document
- The sketch of a document is just a vector of n (say n=200) numbers, which is much easier to deal with than the large set of shingles

Computing Similarity



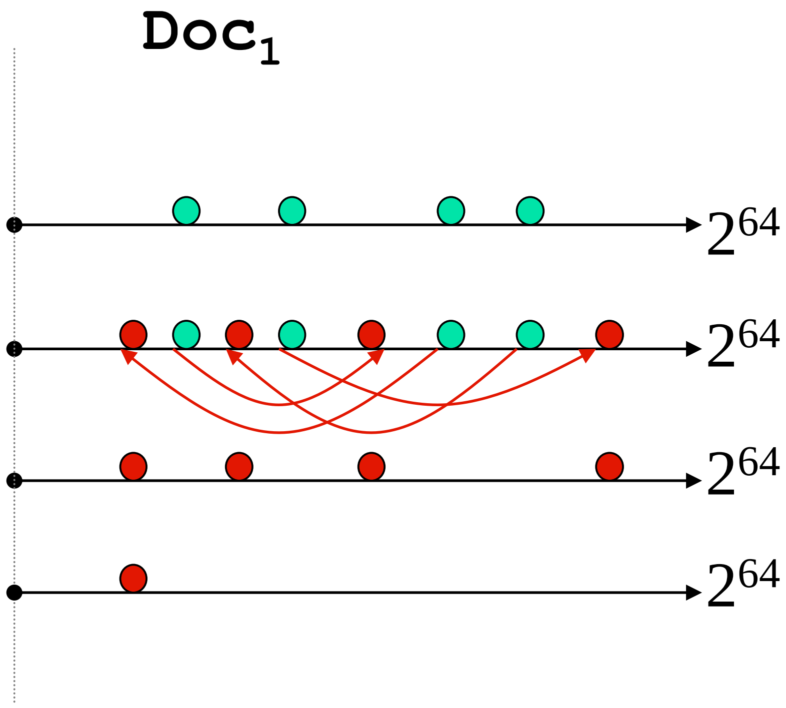
The Jaccard coefficient of two documents can be estimated by the proportion of matching elements in the corresponding pair of sketch vectors

Document Sketch

- For $i = 0 \ldots n$-1
  - Let π_i be a random permutation of all the 2^m possible fingerprints
  - For each document D, its sketch is constructed by setting

$$\boxed{\text{sketch}_D[i] = \min_{s \text{ in } D} \{\ \pi_i(s)\ \}}$$

Document Sketch

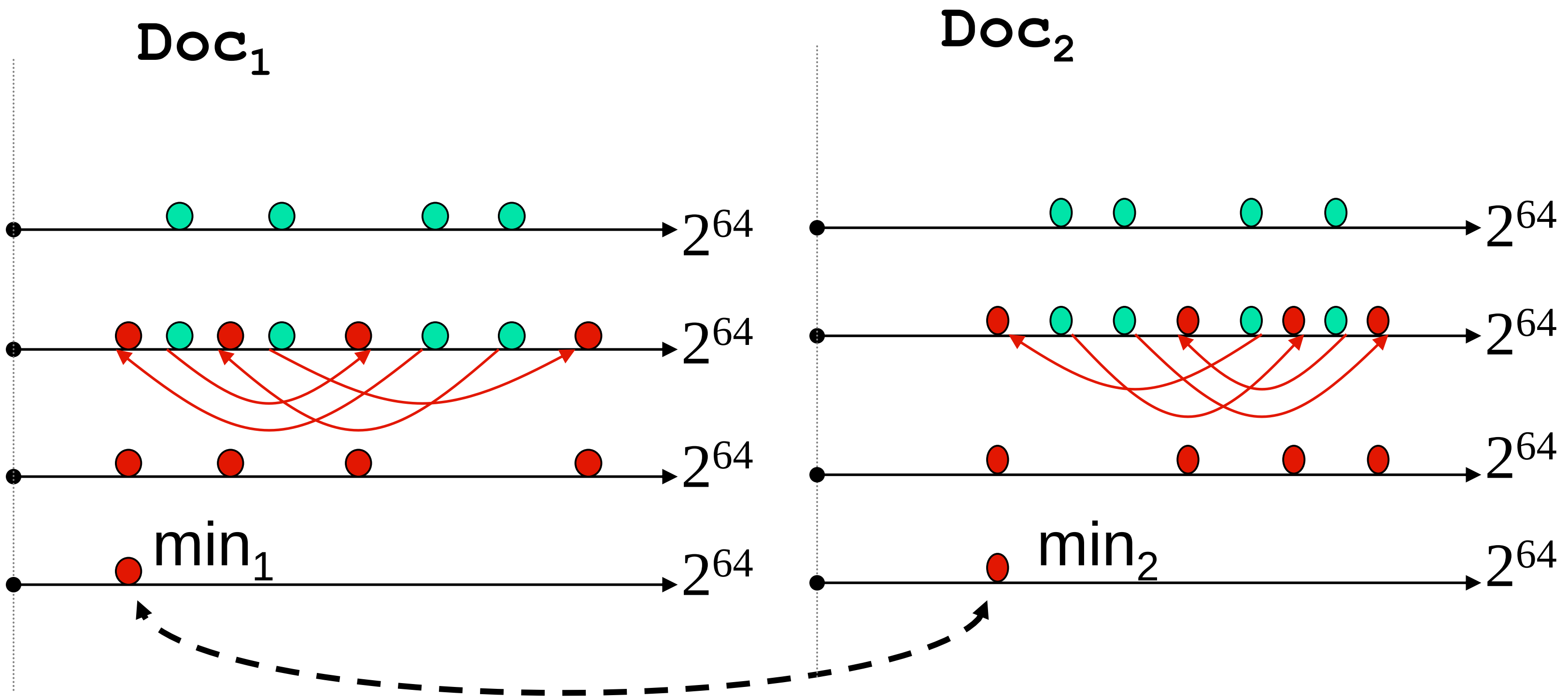


Start with (64-bit) s

Permute on the number line with π_i

Pick the $\min$ value

MinHash



Are these equal?

Check for 200 random permutations: π_1, π_2,... π_{200}

MinHash

- Each random permutation π_i is a test whether Doc_1 and Doc_2 are near-duplicates.
- Every time we see $min_1 = min_2$ we are more confident that they are near-duplicates
- The probability of "matching" permutations where $min_1 = min_2$ actually gives a good estimation for the Jaccard coefficient of Doc_1 and Doc_2

MinHash

- Why?
- Let us view each set of shingles as a column of a matrix A:
  - one row for each element in the universe of 2^m possible shingles.
  - The element $a_{ij} = 1$ indicates the presence of shingle i in set j.

MinHash

- Key Observation
  - There are just four types of rows

	S_{j1}	S_{j2}
C_{11}	1	1
C_{10}	1	0
C_{01}	0	1
C_{00}	0	0

$$\text{Jaccard}(S_{j1}, S_{j2}) = \frac{|S_{j1} \cap S_{j2}|}{|S_{j1} \cup S_{j2}|} = \frac{C_{11}}{C_{01} + C_{10} + C_{11}}$$

MinHash

- For example

S_{j1}	S_{j2}
0	1
1	0
1	1
0	0
1	1
0	1

$$\text{Jaccard}(S_{j1}, S_{j2}) = \frac{|S_{j1} \cap S_{j2}|}{|S_{j1} \cup S_{j2}|} = \frac{2}{5} = 0.4$$

MinHash

- Consider scanning columns $j1$, $j2$ in increasing row index, until the first non-zero entry is found in either column (i.e., "01" or "10" or "11")
- As π_i is a random permutation, the chance that this smallest row has a 1 in both columns (i.e. " 11") is exactly

$$C_{11} / (C_{01}+C_{10}+C_{11})$$

- In other words, the *probability* that $\min_1 = \min_2$ is actually the same as the Jaccard coefficient

MinHash

- This probability estimation from one random permutation is obviously unreliable on its own --- it is always either 0 or 1
- However, it will be fairly accurate when we average over a large number (like n=200) of random permutations.
- Thus, to compute the Jaccard coefficient between two documents, we only need to count the number of “matching” permutations for them and divide it by n=200

MinHash

- Implementation
  - We use a hash functions as an efficient way of doing permutation $\pi_i = h_i : \{0 \ldots 2^m\text{-}1\} \rightarrow \{0 \ldots 2^m\text{-}1\}$
  - Scan all shingles s_k in the union of two sets in arbitrary order
  - For each hash function h_i and documents $D_1, D_2, \ldots$: keep a slot for minimum value found so far
  - If $h_i\,(s_k)$ is lower than the minimum found so far: update the slot

Final Notes

- What we have described is how to detect near-duplicates for a single pair of two documents
- In “real life” we’ll have to concurrently look at many pairs
  - See text book for details
- This family of algorithms for finding similar items is called Locality-Sensitive Hashing (LSH)