Duplicate detection

Kira Radinsky

Based on the Standford slides by Christopher Manning and Prabhakar Raghavan

Duplication

- ~30% of the content on the Web is *near-duplicate* pages
  - Pages with content that is nearly identical to that of other pages
- Issues:
  - Index duplicate content only once
  - Return only one version in the search results
  - How can near-duplicate pages be identified in a scalable and reliable manner?

Duplicate/Near-Duplicate Detection

- *Duplication*: Exact match can be detected with fingerprints
- *Near-Duplication*: Approximate match
  - Compute syntactic similarity with an edit-distance measure
  - Use similarity threshold to detect near-duplicates
    - E.g., Similarity > 80% => Documents are near duplicates
    - Not transitive though sometimes used transitively

Computing Similarity - Shingles

<u>Features for Similarity</u> (Shingles (Word N-Grams))

- Segments of a document (natural or artificial breakpoints)
- K-shingling of a document transforms the document into a set containing all windows of k contiguous terms
- E.g., 4-shingling 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
  }

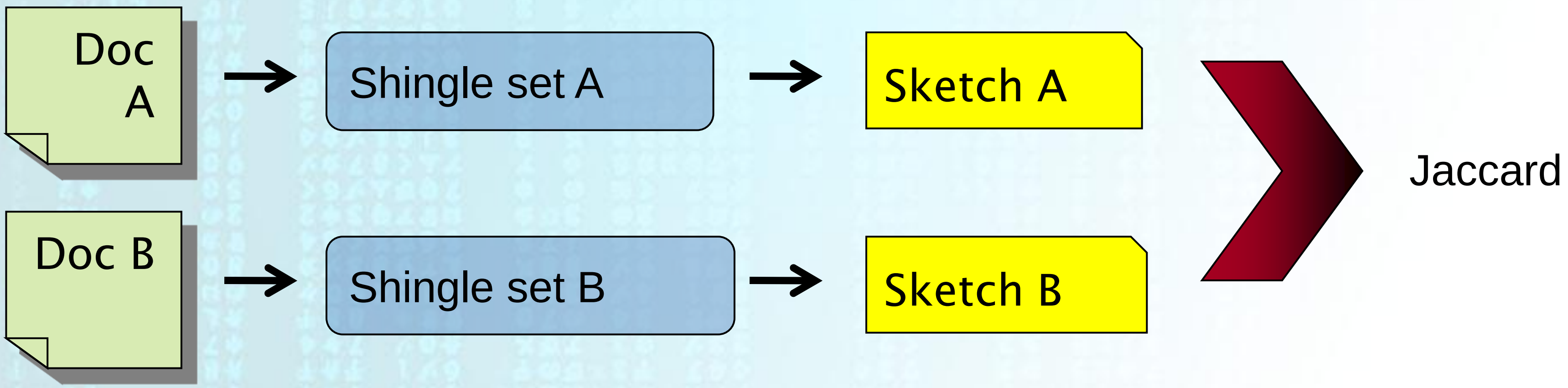
Computing Similarity – distance metric

Similarity Measurement

- Denote by $S_k(d)$ the k-shingling of document d
- Definition: the resemblance of d1 and d2, $R(d1,d2) = |S_k(d1) \cap S_k(d2)| \,/\, |S_k(d1) \cup S_k(d2)|$
- The distance measure $\Delta(d1,d2) = 1\text{-}R(d1,d2)$ is a *metric*

Shingles + Set Intersection

1. Computing <u>exact</u> set intersection of shingles between <u>all</u> pairs of documents is expensive/intractable
   - Approximate using a cleverly chosen subset of shingles from each (a *sketch*)
2. Estimate (size_of_intersection / size_of_union) based on a short sketch



Sketch of a document

Create a sketch vector (of size ~200) for each document

- Documents that share ≥ *t* (usually 80%) corresponding vector elements will be considered near duplicates
- Definitions:
  - Let f map all k-shingles in the universe to $0..2^m$ (e.g., f = fingerprinting)
  - Let π be a *random permutation* on $0..2^m$
- For doc *D*, $sketch_D$ is as follows:
  - Sketch Option 1:

    $F_m(d)=\min_m(\Pi(S_k(d))$ be the m smallest numbers after applying Π to the k-shingling of d
  - Sketch Option 2:

    $V_n(d) = \{ t \in \Pi(S_k(d)) \mid t = 0 \bmod n \}$

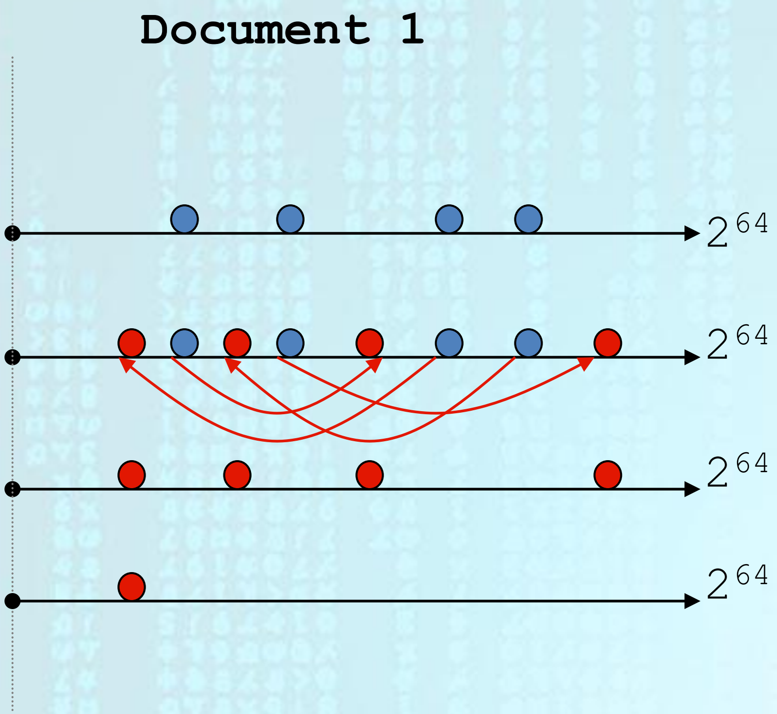
From Shingles to Sketches (cont.)

- A function X is an *unbiased estimator* of a value Y if E(X)=Y
- **Theorem:** when choosing Π u.a.r., the following functions are unbiased estimators of $R(d_1,d_2)$:
  - $| \min_m(F_m(d_1) \cup F_m(d_2)) \cap F_m(d_1) \cap F_m(d_2) | \, / \, | \min_m(F_m(d_1) \cup F_m(d_2)) |$
  - $| V_n(d_1) \cap V_n(d_2) | \, / \, | V_n(d_1) \cup V_n(d_2) |$
- Either $F_m(d)$ or $V_n(d)$ can be chosen as d's sketch
  - $F_m(d)$ has the advantage that it is of fixed size
  - $V_n(d)$ is easier to compute

Multiple Sketches

- Let π_i be a *random permutation* on $0..2^m$
- For doc *D*, $\text{sketch}_D[\, i \,]$ is as follows:
  - Let f map all shingles in the universe to $0..2^m$ (e.g., f = fingerprinting)
  - Let π_i be a *random permutation* on $0..2^m$
  - Pick MIN $\{\pi_i(f(s))\}$ over all shingles *s* in *D*

Computing Sketch[i] for Doc1

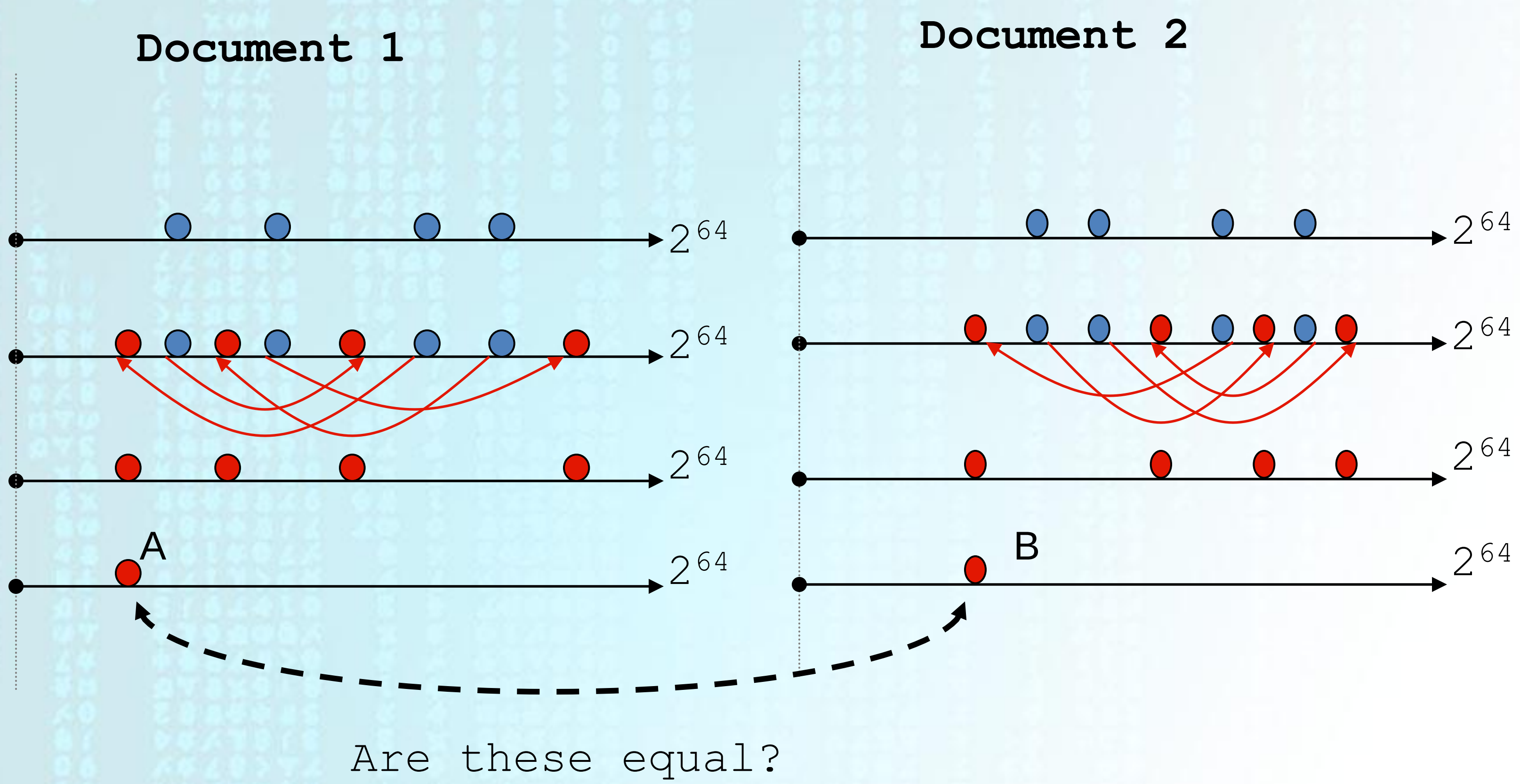


Start with 64-bit *f*(shingles)

Permute on the number line with π_i

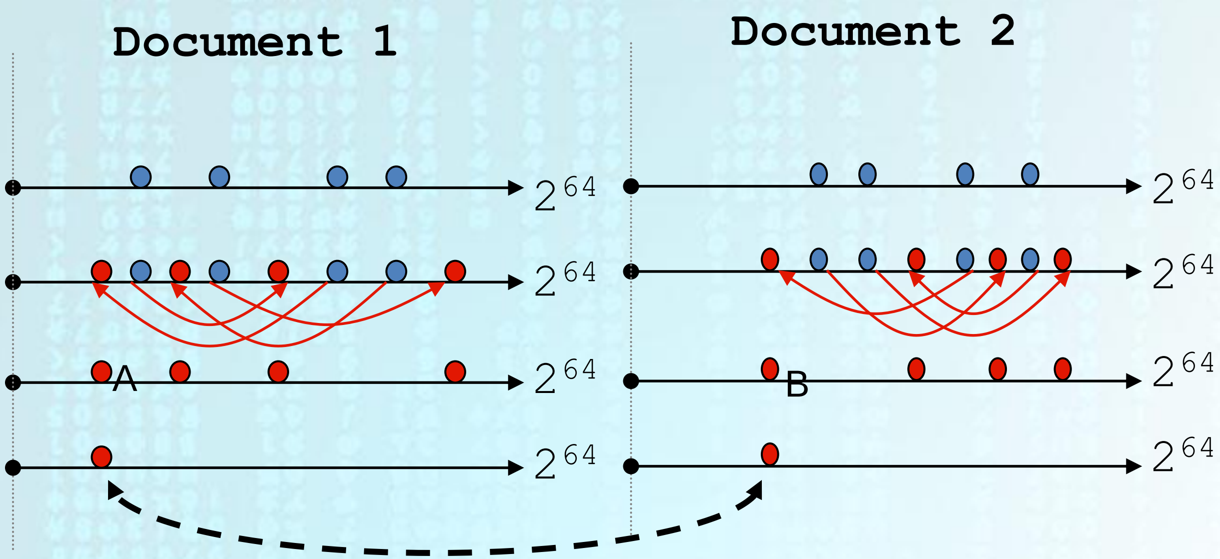
Pick the min value

Test if Doc1.Sketch[i] = Doc2.Sketch[i]



Test for 200 random permutations: $\pi_1, \pi_2, \ldots \pi_{200}$

However…



A = B iff the shingle with the MIN value in the union of Doc1 and Doc2 is common to both (i.e., lies in the intersection)

Claim: This happens with probability

`Size_of_intersection / Size_of_union`

`Why?`

Set Similarity of sets C_i , C_j

$$\mathrm{Jaccard}(C_i, C_j) = \frac{|C_i \cap C_j|}{|C_i \cup C_j|}$$

- View sets as columns of a matrix A
  - one row for each element in the universe.
  - $a_{ij} = 1$ indicates presence of item i in set j
- Example

C_1	C_2
0	1
1	0
1	1
0	0
1	1
0	1

Jaccard(C_1,C_2) **= 2/5 = 0.4**

Key Observation

- For columns C_i, C_j, four types of rows

	C_i	C_j
A	1	1
B	1	0
C	0	1
D	0	0

- Let A = # of rows of type A
- **Claim**

$$\mathrm{Jaccard}(C_i, C_j) = \frac{A}{A + B + C}$$

Min Hashing

- Randomly permute rows
- Hash $h(C_i)$ = index of first row with 1 in column C_i
- Surprising Property

$$P\left[h(C_i)=h(C_j)\right]=\text{Jaccard}\left(C_i,C_j\right)$$

- Why?
  - Both are A/(A+B+C)
  - Look down columns C_i, C_j until first non-Type-D row
  - $h(C_i) = h(C_j)$ ←→ type A row

Min-Hash sketches

- Pick *P* random row permutations
- MinHash sketch

  Sketch_D = list of *P* indexes of first rows with 1 in column C

- Similarity of signatures
  - Let $sim[sketch(C_i),sketch(C_j)]$ = fraction of permutations where MinHash values agree
  - Observe $E[sim(sig(C_i),sig(C_j))] = Jaccard(C_i,C_j)$

Example

Signatures

	S_1	S_2	S_3
Perm 1 = (12345)	1	2	1
Perm 2 = (54321)	4	5	4
Perm 3 = (34512)	3	5	4

	C_1	C_2	C_3
R_1	1	0	1
R_2	0	1	1
R_3	1	0	0
R_4	1	0	1
R_5	0	1	0

Similarities

	1-2	1-3	2-3
Col-Col	0.00	0.50	0.25
Sig-Sig	0.00	0.67	0.00

Algorithm for Clustering Near-Duplicate Documents

1. Compute the sketch of each document
2. From each sketch, produce a list of <shingle, docID> pairs
3. Group all pairs by shingle value
4. For any shingle that is shared by more than one document, output a triplet <smaller-docID, larger-docID, 1> for each pair of docIDs sharing that shingle
5. Sort and aggregate the list of triplets, producing final triplets of the form <smaller-docID, larger-docID, # common shingles>
6. Join any pair of documents whose number of common shingles exceeds a chosen threshold using a "Union-Find" algorithm
7. Each resulting connected component of the UF algorithm is a cluster of near-duplicate documents

Implementation nicely fits the "map-reduce" programming paradigm

Implementation Trick

- Permuting universe even once is prohibitive
- **Row Hashing**
  - Pick P hash functions $h_k: \{1,\ldots,n\} \rightarrow \{1,\ldots,O(n)\}$
  - Ordering under h_k gives random permutation of rows
- **One-pass Implementation**
  - For each C_i and h_k, keep slot for min-hash value
  - Initialize all slot(C_i,h_k) to infinity
  - Scan rows in arbitrary order looking for 1's
    - Suppose row R_j has 1 in column C_i
    - For each h_k,
      - if $h_k(j) <$ slot(C_i,h_k), then slot(C_i,h_k) $\leftarrow h_k(j)$

Example

	C_1	C_2
R_1	1	0
R_2	0	1
R_3	1	1
R_4	1	0
R_5	0	1

$h(x) = x \bmod 5$
$g(x) = 2x+1 \bmod 5$

	C_1 slots	C_2 slots
$h(1) = 1$	1	-
$g(1) = 3$	3	-
$h(2) = 2$	1	2
$g(2) = 0$	3	0
$h(3) = 3$	1	2
$g(3) = 2$	2	0
$h(4) = 4$	1	2
$g(4) = 4$	2	0
$h(5) = 0$	1	0
$g(5) = 1$	2	0