Finding near-duplicate documents

1

Duplicate versus near duplicate documents

- Duplicate = identical?
- Near duplicate:
  small structural differences
    - not just content similarity
- define "small"
  - date change?
  - small edits?
  - metadata change?
  - other?

2

Applications

- Crawling network
- Indexing
- Returning query results
  - cluster near duplicates; return 1
- Plagiarism

3

Framework

- Algorithm to assign quantitative degree of similarity between documents
- Issues
  - What is basic token for documents?
    - character
    - word/term
  - What is threshold for "near duplicate"?
  - What are computational costs?

4

Classic document comparison

- Edit distance
  - count deletions, additions, substitutions to convert Doc_1 into Doc_2
  - each action can have different cost
  - applications
    - UNIX "diff"
    - similarity of genetic sequences
- Edit distance algorithm
  - dynamic programming
  - time O(m*n) for strings length m and n

5

Edit distance for collections

- token = word
  - compare other applications
- Cost is $O(\sum_{i,j} |Doc_i|*|Doc_j|)$
- Right sense of similarity?

6

Addressing computation cost

A general paradigm to find duplicates in N docs:

1. Define function *f* capturing contents of each document in one number
   "Hash function", "signature", "fingerprint"
2. Create <*f*(doc_i), ID of doc_i> pairs
3. Sort the pairs
4. Recognize duplicate or near-duplicate documents as having the same *f* value or *f* values within a small threshold

Compare: computing a similarity score on pairs of documents

7

Optimistic cost

A general paradigm to find duplicates in N docs:

1. Define function *f* capturing contents of each document in one number O(|doc|)
   "Hash function", "signature", "fingerprint"
2. Create <*f*(doc_i), ID of doc_i> pairs $O(\Sigma_{i=1...N} (|doc_i|))$
3. Sort the pairs O(N log N)
4. Recognize duplicate or near-duplicate documents as having the same *f* value or *f* values within a small threshold O(N)

Compare: computing a similarity score on pairs of documents

8

General paradigm: details

1. Define function *f* capturing contents of each document in one number
   "Hash function", "signature", "sketch", "fingerprint"
2. Create <*f*(doc_i), ID of doc_i> pairs
3. Sort the pairs
4. Recognize duplicate or near-duplicate documents as having the same *f* value or *f* values within a small threshold
   - recognize exact duplicates:
     - threshold = 0
     - examine documents to verify duplicates
   - recognize near-duplicates
     Use small "small threshold"
     => "near duplicate" not transitive

9

"Syntactic clustering"

We will look at this one example:

Andrei Z. Broder, Steven C. Glassman, Mark S. Manasse, and Geoffrey Zweig, Syntactic Clustering of the Web
Sixth International WWW Conference, 1997.

- "syntactic similarity" versus semantic
  Sequences of words
- Finding near duplicates
- Doc = sequence of words
  Word = Token
- Uses **sampling**
- Similarity based on **shingles**
- Does compare documents

10

Shingles

- A *w*-shingle is a contiguous subsequence of *w* words
- The *w*-shingling of doc D, S(D, *w*) is the set of ***unique*** w-shingles of D

11

Similarity of docs with shingles

➢ For fixed ***w***, resemblance of docs A and B :
  r(A, B) = |S(A) ∩ S(B)| / |S(A) U S(B)|
  Jaccard coefficient

- For fixed ***w***, containment of doc A in doc B :
  C(A, B) = |S(A) ∩ S(B)| / |S(A)|
- For fixed ***w***, resemblance distance betwn docs A and B :
  D(A, B) = 1- r(A, B)
  Is a metric (triangle inequality)

Note we are now comparing documents!

12

Example

A: "a rose is red a rose is white"

4-shingles:

"a rose is red"
"rose is red a"
"is red a rose"
"red a rose is"
"a rose is white"

B: "a rose is white a rose is red"

4-shingles:

"a rose is white"
"rose is white a"
"is white a rose"
"white a rose is"
"a rose is red"

r(A, B) = 0.25

13

Compare

A: "a rose is red a rose is white"

3-shingles:

"a rose is"
"rose is red"
"is red a"
"red a rose"
"rose is white"

B: "a rose is white a rose is red"

3-shingles:

"a rose is"
"rose is white"
"is white a"
"white a rose"
"rose is red"

r(A, B) = 0.43

14

Sample of shingles

Want to ***estimate*** r and/or c

Do this by calculating approximation on a sample of shingles **for fixed *w***

- 1-to-1 map each shingle to integer in fixed, large range R
  - 64-bit hash, R=[0, 2^{64}-1]
- Let Π be a random permutation from R to R
- For any S(D) define:

  H(D) = Set of integer hash values corresponding to shingles in S(D)

  Π(D) = Set of permuted values in H(D)

  ***x*(Π, D) = smallest integer in Π(D)**

15

Sketch of shingles

- Let $\Pi_1, \ldots, \Pi_m$ be m random permutations R → R
  - text: m=20

The sketch of doc D for $\Pi_1, \ldots, \Pi_m$ is

$\psi(D) = \{x(\Pi_i, D) \mid 1 \le i \le m\}$

doc → set shingles → set integers

→ m sets permuted integers

→ m smallest integers: one per permutation

Sketch is a ***sampling***

16

Approximation of resemblance

Theorem:

For random permutation Π:

r(A, B) = P ($x(\Pi, A) = x(\Pi, B)$)

Estimate P ($x(\Pi, A) = x(\Pi, B)$) as

$| \psi(A) \cap \psi(B) | / m$

recall m is # permutations

17

Last time

- Defining and detecting near-duplicate documents

Today

- Example Syntactic Clustering algorithm
- Finish detecting near-duplicates
- Non-text retrieval

Next

- Finish non-text retrieval

18

Example: compare

A: "a rose is red a rose is white"

3-shingles:

1"a rose is"

2"rose is red"

3"is red a"

4"red a rose"

5"rose is white"

B: "a rose is white a rose is red"

3-shingles:

"a rose is" 1

"rose is white" 5

"is white a" 6

"white a rose" 7

"rose is red" 2

r(A, B) = 0.43

19

Example mappings

- R = [0, 10000]
- Let H(i) = i*1000; $1 \le i \le 7$
- Let m=5
- Define a permutation
  - Example
    - Get randval = Math.random()
    - Compute function of randval and H(i) to get $\Pi(i)$
- Do 5 times for 5 permutations

20

$\psi(A) = \{x(\Pi_i, A) \mid 1 \le i \le m\} = \{568, 1150, 6119, 6880, 1905\}$

Π_1:	Π_2:	Π_3:	Π_4:	Π_5:
568	1150	9223	9376	2976
1136	2301	8447	8752	5952
1705	3452	7671	8128	8929
2273	4602	6895	7504	1905
2842	5753	6119	6880	4881
3410	6904	5343	6256	7858
3979	8054	4567	5633	834

21

$\psi(B) = \{x(\Pi_i, B) \mid 1 \le i \le m\} = \{568, 1150, 4567, 5633, 834\}$

Π_1:	Π_2:	Π_3:	Π_4:	Π_5:
568	1150	9223	9376	2976
1136	2301	8447	8752	5952
1705	3452	7671	8128	8929
2273	4602	6895	7504	1905
2842	5753	6119	6880	4881
3410	6904	5343	6256	7858
3979	8054	4567	5633	834

22

$\psi(A) = \{x(\Pi_i, A) \mid 1 \le i \le m\} = \{568, 1150, 6119, 6880, 1905\}$

$\psi(B) = \{x(\Pi_i, B) \mid 1 \le i \le m\} = \{568, 1150, 4567, 5633, 834\}$

Π_1:	Π_2:	Π_3:	Π_4:	Π_5:
568	1150	9223	9376	2976
1136	2301	8447	8752	5952
1705	3452	7671	8128	8929
2273	4602	6895	7504	1905
2842	5753	6119	6880	4881
3410	6904	5343	6256	7858
3979	8054	4567	5633	834

Resemblance estimate:
$|\psi(A) \cap \psi(B)| / m$
$= 2/5 = .4$

Actual resemblance
$= 3/7 = .43$

23

Algorithm used (text's version)

1. Calculate *sketch* $\psi(D_i)$ for every doc D_i
2. Calculate $|\psi(D_i) \cap \psi(D_j)| = ct_{ij}$ for each non-empty intersection:
   i. Produce list of <shingle value, docID> pairs for all shingle values $x(\Pi_k, D_i)$ in the sketch for each doc.
   ii. Sort the list by shingle value
   iii. Produce all triples <ID(D_i), ID(D_j), $ct_{i,j}$> for which $ct_{i,j}$>0
   This *not linear-time* for the list of docs for one shingle value
3. Recognize duplicate, near-duplicate documents: resemblance $ct_{i,j}/m$ above a large threshold

24

Algorithm cost

1. Calculate *sketch* $\psi(D_i)$ for every D_i $O(\Sigma_i m|D_i|)$
2. Calculate $|\psi(D_i) \cap \psi(D_j)| = ct_{ij}$ for each non-empty intersection:
   i. Produce list of <shingle value, docID> pairs for all shingle values $x(\Pi_k, D_i)$ in the sketch for each doc.
   ii. Sort the list by shingle value
   iii. Produce all triples <ID(D_i), ID(D_j), $ct_{i,j}$> for which $ct_{i,j}>0$
   This *not linear-time* for the list of docs for one shingle value
3. Recognize duplicate, near-duplicate documents: resemblance $ct_{i,j}/m$ above a large threshold

Algorithm cost

1. Calculate *sketch* $\psi(D_i)$ for every D_i $O(\Sigma_i m|D_i|)$
2. Calculate $|\psi(D_i) \cap \psi(D_j)| = ct_{ij}$ for each non-empty intersection:
   i. Produce list of <shingle value, docID> pairs for all shingle values $x(\Pi_k, D_i)$ in the sketch for each doc.
   ii. Sort the list by shingle value $O(mN \log (mN))$
   iii. Produce all triples <ID(D_i), ID(D_j), $ct_{i,j}$> for which $ct_{i,j}>0$
   This *not linear-time* for the list of docs for one shingle value
3. Recognize duplicate, near-duplicate documents: resemblance $ct_{i,j}/m$ above a large threshold

Algorithm cost

1. Calculate *sketch* $\psi(D_i)$ for every D_i $O(\Sigma_i m|D_i|)$
2. Calculate $|\psi(D_i) \cap \psi(D_j)| = ct_{ij}$ for each non-empty intersection:
   i. Produce list of <shingle value, docID> pairs for all shingle values $x(\Pi_k, D_i)$ in the sketch for each doc.
   ii. Sort the list by shingle value $O(mN \log (mN))$
   iii. Produce all triples <ID(D_i), ID(D_j), $ct_{i,j}$> for which $ct_{i,j}>0$
   This *not linear-time* for the list of docs for one shingle value $O(mN^2)$
3. Recognize duplicate, near-duplicate documents: resemblance $ct_{i,j}/m$ above a large threshold

Algorithm cost

1. Calculate *sketch* $\psi(D_i)$ for every D_i $O(\Sigma_i m|D_i|)$
2. Calculate $|\psi(D_i) \cap \psi(D_j)| = ct_{ij}$ for each non-empty intersection:
   i. Produce list of <shingle value, docID> pairs for all shingle values $x(\Pi_k, D_i)$ in the sketch for each doc.
   ii. Sort the list by shingle value $O(mN \log (mN))$
   iii. Produce all triples <ID(D_i), ID(D_j), $ct_{i,j}$> for which $ct_{i,j}>0$
   This *not linear-time* for the list of docs for one shingle value $O(mN^2)$
3. Recognize duplicate, near-duplicate documents: resemblance $ct_{i,j}/m$ above a large threshold $O(N^2)$

Revisit the original paradigm

A general paradigm to find duplicates in N docs:

1. Define function *f* capturing contents of each document in one number $O(|doc|)$
   "Hash function", "signature", "fingerprint"
2. Create <*f*(doc_i), ID of doc_i> pairs $O(\Sigma_{i=1...N} (|doc_i|))$
3. Sort the pairs $O(N \log N)$
4. Recognize duplicate or near-duplicate documents as having the same *f* value or *f* values within a small threshold $O(N)$

Compare: computing a similarity score on pairs of documents

Syntactic Clustering Paradigm

- Does compare docs, so not same as paradigm we started with, but uses ideas
- Contents of doc captured by sketch – a set of shingle values
- Similarity of docs scored by count of common shingle values for docs
- Don't look at all doc pairs, look at all doc pairs that share a shingle value
- Text clusters by similarity threshold

More efficient : supershingles

"meta-sketch"

1. Sort shingle values of a sketch
2. Compute the shingling of the sequence of shingle values
    - Each original shingle value now a token
    - Gives "supershingles"
3. "meta-sketch" = set of supershingles

One supershingle in common =>
sequences of shingles in common
Documents with ≥1 supershingle in common => similar

- Each supershingle for a doc. characterizes the doc
- Sort <supershingle, docID> pairs: docs sharing a supershingle are similar => our first paradigm

31

Pros and Cons of Supershingles

\+ Faster
- Problems with small documents – not enough shingles
- Can't do containment

Shingles of superset that are not in subset break up sequence of shingle values

32

Using with Web Crawling

- Want know if new doc. too similar to ones seen
- What this calculation look like?

33

Using with Web Crawling

- Want know if new doc. too similar to ones seen
- No clustering required
- calculate sketch or supershingle of new document
- Look up to see if have similar document
    – or similar document that is fresh enough
    – Need efficient look-up

34

Variations of shingling

- Can define different ways to do sampling
- Studies in original paper used modular arithmetic
    – sketch formed by taking shingle hash values mod some selected m

35

Original experiments (1996) by Broder et. al.

- 30 million HTML and text docs (150GB) from Web crawl
- 10-word shingles
- 600 million shingles (3GB)
- 40-bit shingle "fingerprints"
- Sketch using 4% shingles (variation of alg. we've seen)
- Used count of shingles for similarity
- Using threshold *t* = 50%, found
    – 3.6 million clusters of 12.3 million docs
    – 2.1 million clusters of identical docs – 5.3 million docs
    – remaining 1.5 million clusters mixture: "exact duplicates and similar"

36