Finding near-duplicate documents

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?

Applications

• creating collection
  – indexing
• Crawling network
• Returning query results
  – cluster near duplicates; return 1
• Plagiarism

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?

Classic document comparison

• Edit distance
  – count deletions, additions, substitutions to convert Doc\textsubscript{1} into Doc\textsubscript{2}
  – can each action can have different cost
  – applications
    • UNIX “diff”
    • similarity of genetic sequences
• Edit distance algorithm
  – dynamic programming
  – time $O(|Doc_1||Doc_2|)$

Edit distance for collections

• token = word
  – compare other applications
• Cost is $O(\sum |Doc_i||Doc_j|)$
• Right sense of similarity?
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), ID of doc> 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

Optimistic 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), ID of doc> 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

General paradigm: details

1. Define function f capturing contents of each document in one number
   “Hash function”, “signature”, “sketch”, “fingerprint”
2. Create <f(doc), ID of doc> 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
     Problem with “small threshold”?

General paradigm: details

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
     Problem with “small threshold”?

How deal with
<1, D_1> <1.01, D_2> <1.02, D_3> .....<1.99, D_{100}>
and threshold .01 (using ≤ threshold) ?

“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

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
Similarity of docs with shingles

- For fixed $w$, resemblance of docs $A$ and $B$:
  
  $r(A, B) = \frac{|S(A) \cap S(B)|}{|S(A) \cup S(B)|}$

  Jaccard coefficient

- For fixed $w$, containment of doc $A$ in doc $B$:
  
  $C(A, B) = \frac{|S(A) \cap S(B)|}{|S(A)|}$

- For fixed $w$, resemblance distance between docs $A$ and $B$:
  
  $D(A, B) = 1 - r(A, B)$

  Is a metric (triangle inequality)

  Note we are now comparing documents!

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"

$D(A, B) = 0.25$

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"

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 $\Pi$ be a random permutation from $R$ to $R$

- For any $S(D)$ define:
  
  $H(D) =$ Set of integer hash values corresponding to shingles in $S(D)$

  $\Pi(D) =$ Set of permuted values in $H(D)$

  $x(\Pi, D) =$ smallest integer in $\Pi(D)$

Sketch of shingles

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

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

  $\psi(D) = \{x(\Pi_i, D) | 1 \leq i \leq m\}$

  doc $\rightarrow$ set shingles $\rightarrow$ set integers

  $\rightarrow$ $m$ sets permuted integers

  $\rightarrow$ $m$ smallest integers: one per permutation

  Sketch is a sampling

Approximation of resemblance

Theorem:

For random permutation $\Pi$:

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

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

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

recall $m$ is # permutations

Algorithm used (text's version)

1. Calculate sketch $\psi(D)$ for every doc $D_i$

2. Calculate $|\psi(D_i) \cap \psi(D_j)| = c_{ij}$ for each non-empty intersection:

   i. Produce list of $<$shingle value, docID$>$ pairs for all shingle values $x(\Pi, D)$ in the sketch for each doc.

   ii. Sort the list by shingle value

   iii. Produce all triples $<$id(D), id(D'), $c_{ij}>$ for which $c_{ij} > 0$

   This not linear-time for the list of docs for one shingle value

3. Build clusters of similar/almost identical docs

Degree of similarity depends on threshold …
**Clustering**

1. Define docs to be similar if approximate resemblance greater than a predetermined threshold $t$:
   
   $$\frac{c_{ij}}{m} > t$$

2. Build graph of docs: edge between each pair of similar docs

3. The clusters of similar docs are the connected components in the graph
   
   - what type 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
- Uses clustering by similarity threshold

**Algorithm cost**

1. Calculate sketch $\psi(D_i)$ for every $D_i$ $O(|D_i|)$

2. Calculate $|\psi(D_i) \cap \psi(D_j)| = c_{ij}$ for each non-empty intersection:
   - i. Produce list of <shingle value, docID> pairs for all shingle values $x(\psi(D_i))$ 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_{ij} > for which $ct_{ij} > 0$
     
     This not linear-time for the list of docs for one shingle value $O(mN^2)$

3. Build clusters of similar/almost identical docs
   
   Degree of similarity depends on 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

**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), ID of doc> pairs $O(\sum_{i=1...N} (|doc_i|))$
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

**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
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

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 $m$

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*