**Similarity**

CPS 296.3: Information Management and Mining
Jun Yang
Duke University
February 3, 2008

† Thanks to contents borrowed from Ullman (http://infolab.stanford.edu/~ullman/mining/mining.html)

---

**Announcements**

- Office hours: Mondays and Tuesdays 4-5pm
- Seminar-style (reading/discussion) portion of the course will start in three weeks

---

**Example: face recognition**

- Given a large database of face images (say 1 million), find the most similar faces in the database
- Each image represented by a large number (say 1000) numerical features
  - E.g., some (relatively invariant) value like ratio of nose width to eye width
- Two images are similar if at least some fraction (say ¾) of the features are close
- Many-one problem: given a new face, see if it is close to any of the 1 million old faces
- Many-many problem: find which pairs of the 1 million faces are similar

---

**A simple solution**

- Represent each face by a 1000-vector
- Score similarity by comparing pairs of vectors
- Sort-of okay for many-one problem
- Out of the question for many-many problem
  - \( \approx 10^6 \times 10^6 \times 1000 / 2 \) numerical comparisons!

---

**Example 2: entity resolution**

- Given two large sets (say 1 million each) of name-address-phone records, find pairs (one from each set) representing the same person
- Errors of many kinds
  - Typos, missing middle initial, area-code changes, St./Street, Bob/Robert, etc.
- Choose a similarity function between two records
  - E.g., for names, deduct so much for edit distance > 0, so much for missing middle initial, etc.
  - Similarly score addresses and phone numbers
- Sufficiently high total score \( \rightarrow \) records represent the same entity
Example 3: Mining purchases
- Common pattern: look for sets with relatively large intersection
  - E.g., represent a Netflix customer by the set of movies she rented
    - Similar customers have relatively large fraction of their choices in common
  - E.g., represent a product by the set of customers who purchased it
    - Similar movies have many renters in common
  - Tricky: Sony & Samsung TV’s are “similar,” but not typically bought by the same customers

Example 4: Similar documents
- Given a body of docs, e.g., the Web, find pairs of docs that have a lot of text in common, e.g.:
  - Mirrors, or approximate mirrors
  - Plagiarism, including large quotations
  - Repetitions of news articles
- Challenge: how to represent a doc so it is easy to compare with others
  - Special cases are easy (how?), e.g., identical docs, or doc contained verbatim in another
  - General case is hard, when many small pieces of one doc appear out of order in another

Roadmap
- Similar customers
  - Sets or Boolean vectors
    - Technique: Minhashing
      - Signatures
  - Similar products
    - Technique: Shingling

Representing docs
- Represent a doc by its set of shingles (k-grams)
- Summarize the shingle set by a signature—a small piece of data with the property that similar docs are very likely to have “similar” signatures
- Then, the doc similarity problem becomes the problem of finding similar sets

Shingles
- A k-shingle (k-gram) for a doc is a sequence of k characters that appears in the doc
  - Example: k = 2; doc = abcab
    - Set of 2-shingles = {ab, bc, ca}
    - Another option is to regard shingles as a “bag,” and count ab twice
  - Long shingles are typically compressed—hashed to (say) 4 bytes
    - Doc = set of hash values of its k-shingles
    - Two docs could (rarely) appear to have shingles in common, when in fact the hash values collided

Next: Minhashing
- Many similarity problems can be couched as finding subsets of some universal set that have large intersection, e.g.:
  - Docs as sets of shingles (or hashes of these shingles)
  - Similar customers/products
  - Minhashing: a way of compressing sets into signatures such that signature similarity approximates set similarity
From sets to a Boolean matrix

- Rows = elements of the universal set
- Columns = sets (subsets of the universal set)
- 1 in the row for element \( e \) and the column for set \( S \) iff \( e \in S \)

\[
\begin{array}{c|cccc}
S & T & U & V & W \\
\hline
a & 1 & 1 & 0 & 1 & 0 \\
b & 1 & 0 & 1 & 1 & 0 \\
c & 1 & 0 & 0 & 1 & 0 \\
d & 0 & 1 & 0 & 0 & 1 \\
e & 1 & 0 & 1 & 0 & 1 \\
f & 1 & 1 & 0 & 1 & 1 \\
g & 0 & 1 & 0 & 1 & 1 \\
h & 0 & 1 & 0 & 1 & 0 \\
\end{array}
\]

Set Similarity

- Jaccard similarity between two sets: ratio of the sizes of their intersection and union
- \( \text{Sim}(S_1, S_2) = \frac{|S_1 \cap S_2|}{|S_1 \cup S_2|} \)

Recall: a column = set of rows in which it has 1

\[
\begin{align*}
\text{Example: } & C_1, C_2 \\
\text{Sim}(C_1, C_2) &= \frac{2}{5} = 0.4
\end{align*}
\]

Approach outline

- Compute signatures of columns
  - Read from disk to main memory
  - Find similar signatures in main memory
    - Critical: similarity of signatures \( \approx \) similarity of columns
  - Optional: check that columns with similar signatures are really similar
    - To remove false positives

Signatures

- Key idea: “hash” column \( C \) to a small \( \text{Sig}(C) \), such that
  - \( \text{Sig}(C) \) is small enough that all column signatures can fit in main memory
  - \( \text{Sim}(C_i, C_j) \) is the same as the “similarity” of \( \text{Sig}(C_i) \) and \( \text{Sig}(C_j) \)

Does the following approach work?

- Pick 100 rows at random, and let \( \text{Sig}(C) \) be the 100 bits of \( C \) in these rows

\[\text{No! Sparse matrix } \Rightarrow \text{ most signatures will have all 0's, even though the columns have different 1's elsewhere}\]

Minhashing

- Imagine the rows are permuted randomly
- Define “(min)hash” function \( h(C) \) = the index of the first (in the permuted order) row in which column \( C \) has 1

\[\text{Use several (100?) independent hash functions to create a signature (of 100 integers)}\]

Warnings

- Comparing all pairs of signatures may take too much time, even if not too much space
  - A job for Locality-Sensitive Hashing; more later
- These methods can give false negatives
  - And even false positives, if the optional check on the last slide is skipped
Minhashing example

<table>
<thead>
<tr>
<th>Input matrix</th>
<th>Signature matrix</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 0 1 0</td>
<td>2 1 1 1</td>
</tr>
<tr>
<td>1 0 0 1</td>
<td>2 1 1 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>2 1 1 1</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>0 1 0 1</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>1 1 0 0</td>
</tr>
<tr>
<td>1 0 1 0</td>
<td>1 1 0 0</td>
</tr>
</tbody>
</table>

Surprising property

- Given columns $C_1$ and $C_2$, rows can be classified into four types:
  \[ C_1 \quad C_2 \]
  \[
  \begin{array}{ll}
  1 & 1 \\
  1 & 0 \\
  0 & 1 \\
  0 & 0 
  \end{array}
  \]
- Let $n =$ # of rows, and $n_{xx} =$ # of rows of type $xx$
- $\text{Sim}(C_1, C_2) = \frac{n_{11}}{n_{11} + n_{10} + n_{01}}$
- The probability (over all permutations of rows) that $h(C_1) = h(C_2)$ is the same as $\text{Sim}(C_1, C_2)$

Why?

- # of permutations for which $h(C_1) = h(C_2) = i$:
  \[ P(n_{10}, i - 1) n_{11} (n - i)! \]
- # of permutations for which the first 1 in either $C_1$ or $C_2$ (or both) shows up at index $i$:
  \[ P(n_{10}, i - 1) (n_{11} + n_{10} + n_{01}) (n - i)! \]
- The ratio is exactly $n_{11} / (n_{11} + n_{10} + n_{01})$
- A simpler proof?
  - Consider $\Pr(h(C_1) = h(C_2) \mid \text{first 1 shows up at index } i)$

Similarity of signatures

- The similarity between two signatures is the fraction of the rows in which they agree
- Recall each signature row $k$ corresponds to a hash function $h_k$
- $\text{Sim}(\text{Sig}(C_1), \text{Sig}(C_2)) = \sum_k 1(h_k(C_1) = h_k(C_2)) / K$

Implementation challenges

- Suppose the size of the universal set is 1 billion
- Difficult to pick a random permutation of 1 billion
- Representing a random permutation requires 1 billion entries
- Access rows in permuted order leads to thrashing

Ideas

- Instead of truly random permutations, settle for something close to a min-wise independent family of permutations
  - [Broder et al. STOC '98]
  - i.e., if we randomly pick $\pi$ from this family, then for any $C$, all its elements have an equal chance to become the first one after applying $\pi$
  - E.g., pick (say) 100 hash functions of the form $\pi(x) = ax + b \mod p$ ($p$ is the universe size and prime)
  - Instead of making the outer loop go over every hash function, scan the Boolean matrix once and process all hash functions
Implementation

keep signature matrix $M$ in memory
for each row $r$ corresponding to an element
for each column $c$ corresponding to a set
if $c$ has 1 in row $r$
for each permutation $\pi_k$
if $\pi_k(r) < M(k, c)$ then
$M(k, c) \leftarrow \pi_k(r)$

Some more details

• If the input (Boolean matrix) is stored row-by-row, then only one pass is needed

• If the input is stored column-by-column
  - E.g., doc by doc
  - Option 1: represent it by (row, column) pairs and sort it once by row
    - External-memory merge sort is pretty scalable
  - Option 2?

“Sufficiently similar”

• For finding similar sets
  - Pick a threshold $s (< 1)$ for Jaccard similarity
  - A pair of columns $C_1$ and $C_2$ is a candidate pair if their minhash signatures agree in at least fraction $s$ of the rows
• For images, a pair of vectors is a candidate if they differ by at most a small amount $t$ in at least $s$ fraction of the components
• For entity records, a pair is a candidate if the sum of similarity scores of corresponding components exceeds a threshold

Example

<table>
<thead>
<tr>
<th>Row</th>
<th>$C_1$</th>
<th>$C_2$</th>
<th>Sig1</th>
<th>Sig2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>$\pi_1(1) = 1$</td>
<td>$\pi_1(1) = 3$</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>$\pi_2(1) = 3$</td>
<td>$\pi_2(1) = 1$</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>$\pi_2(2) = 1$</td>
<td>$\pi_2(2) = 2$</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>$\pi_2(3) = 0$</td>
<td>$\pi_2(3) = 3$</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>$\pi_2(4) = 4$</td>
<td>$\pi_2(4) = 2$</td>
</tr>
</tbody>
</table>

Finding similar pairs

• Suppose we have in main memory data representing a large number of objects, e.g.:
  • Feature vectors for face images
  • Minhash signatures for docs
• We want to compare each to each, finding those pairs that are sufficiently similar

“Sufficiently similar”

• For finding similar sets
  • Pick a threshold $s (< 1)$ for Jaccard similarity
  • A pair of columns $C_1$ and $C_2$ is a candidate pair if their minhash signatures agree in at least fraction $s$ of the rows
• For images, a pair of vectors is a candidate if they differ by at most a small amount $t$ in at least $s$ fraction of the components
• For entity records, a pair is a candidate if the sum of similarity scores of corresponding components exceeds a threshold

Checking all pairs is hard

• E.g., $10^6$ columns $\Rightarrow 5 \times 10^{11}$ comparisons, or 6 days at 1 $\mu$s /comparison

Solutions

• Divide-Compute-Merge (DCM) uses external-memory sorting, merging
• Locality-Sensitive Hashing (LSH) can be carried out in main memory, but admits false negatives
**Divide-compute-merge**

- Designed for problems where data is presented by column (e.g., shingles and docs)
- At each stage, divide data into batches that fit in main memory
- Operate on individual batches and write out partial results to disk
- Merge partial results from disk

**DCM steps**

![DCM steps diagram](image)

**DCM notes**

- Streamline adjacent steps
  - E.g., the first invert step can be combined with the first pass of sorting on shingleId
  - In multi-pass sort-merge, eagerly merge/aggregate
  - Eliminate very common shingles
  - Eliminate exact-duplicate docs first

**Locality-sensitive hashing**

- Hash columns of signature matrix \( M \) several times
- Arrange that (only) similar columns are likely to hash to the same bucket
- Candidate pairs are those that hash at least once to the same bucket

**Partitioning into bands**

- Divide matrix \( M \) into \( b \) bands of \( r \) rows each
- For each band, hash its portion of each column to a hash table
- Candidate column pairs are those that hash to the same bucket for at least one band

* Tips: Tune \( b \) and \( r \) to catch similar pairs, but few dissimilar pairs

**Hashing by band illustrated**

![Hashing by band illustrated diagram](image)
A simplifying assumption

- There are enough number of buckets that columns are unlikely to hash to the same bucket unless they are identical in particular band.
- Hereafter, we assume “same bucket” = “identical for the band”.

Example

- 100,000 columns, 100 integer signature rows
  - \( M \) takes 40MB
  - But 5,000,000,000 pairs of signatures take a while to compare
- Choose 20 bands with 5 integers each
- Suppose \( C_1, C_2 \) are 80% similar
  - \( \Pr[C_1, C_2 \text{ identical in a given band}] = (0.8)^2 = 0.328 \)
  - \( \Pr[C_1, C_2 \text{ not identical in any band}] = (1 - 0.328)^{20} = 0.00035 \)
  - I.e., we miss \( \approx 1/3000 \) of the 80%-similar column pairs
- Suppose \( C_1, C_2 \) are only 40% similar
  - \( \Pr[C_1, C_2 \text{ identical in a given band}] = (0.4)^5 = 0.01 \)
  - \( \Pr[C_1, C_2 \text{ identical in at least one band}] = 1 - (1 - 0.01)^{20} = 0.18 \)

LSH involves a tradeoff

- Pick # of bands and # of rows per band (product = # of minhashes) to balance false positives/negatives.
- Example: if we had fewer than 20 bands, the number of false positives would go down, but the number of false negatives would go up.
  - Why?

Analysis of LSH: what we want

<table>
<thead>
<tr>
<th>Probability of sharing a bucket</th>
<th>Similarity of two columns</th>
</tr>
</thead>
<tbody>
<tr>
<td>No chance if ( s &lt; t )</td>
<td></td>
</tr>
<tr>
<td>Probability = 1 if ( s &gt; t )</td>
<td></td>
</tr>
</tbody>
</table>

What testing one row offers

- Probability of sharing a bucket = similarity

What \( b \) bands of \( r \) rows offer

- \( \Pr[\text{a band identical}] = \frac{(1/b)^b}{1 - (1 - \frac{1}{b})^b} \)
- \( \Pr[\text{a band unequal}] = 1 - \frac{(1/b)^b}{1 - (1 - \frac{1}{b})^b} \)
- At least one band identical
  - All rows of a band are equal
  - No bands identical
  - Some row of a band unequal
LSH summary

- Tune to get almost all pairs with similar signatures, but eliminate most pairs with dissimilar signatures
- In main memory, check that candidate pairs really do have similar signatures
- Optional: in another pass through data, check that the remaining candidate pairs really are similar columns in the input Boolean matrix

LSH for other applications

- Face recognition from 1000 measurements/face
- Entity resolution from name-address-phone records

General approach: find many hash functions for elements; candidate pairs share a bucket for at least one hash function

Face-rec. “hash” functions

- Pick a set of \( r \) of the 1000 features
- Each bucket corresponds to a range of values for each of the \( r \) selected features
- Map a vector to the bucket such that all its \( r \) selected components are in range
- Optional: if near the edge of a bucket, also map to an adjacent bucket

Example: \( r = 2 \)

- Given an image (the probe), map it to buckets using all functions
- Any member of any one of its buckets is a candidate
- For each candidate, count the # components in which the candidate and probe are close
- Match if # of close components \( \geq \) threshold

Many-one face lookup

- As before, use many different “hash” functions
  - Each based on a different set of the 1000 features
  - Each bucket of each function points to images that map to that bucket

- Given an image (the probe), map it to buckets using all functions
- Any member of any one of its buckets is a candidate
- For each candidate, count the # components in which the candidate and probe are close
- Match if # of close components \( \geq \) threshold

Hashing the probe

- Look in all these buckets

Example:

<table>
<thead>
<tr>
<th>Bucket</th>
<th>Range</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-4</td>
<td></td>
</tr>
<tr>
<td>5-9</td>
<td></td>
</tr>
<tr>
<td>10-14</td>
<td></td>
</tr>
<tr>
<td>15-19</td>
<td></td>
</tr>
</tbody>
</table>

One bucket, for \((x, y)\):

- if \(10 \leq x \leq 16\) and \(0 \leq y \leq 4\)
- \((27,9)\) goes here
- Maybe put a copy here, too

Hashing the probe

- Look in all these buckets
Many-many problem
• Make each pair of images that are in the same bucket (according to any hash function) be a candidate
• Score each candidate pair as for the many-one problem

Matching customer records
Example from Ullman’s consulting experience
• A agreed to solicit customers for B, for a fee
• They then had a parting of the ways, and argued over how many customers
• Neither recorded which customers were involved
• B had ~1 million records of all its customers
• A had ~1 million records on customers, some of whom it had signed up for B
• Records had name, address, and phone, but for various reasons, they could be different for the same person

LSH-inspired approach
• Three hash functions: on name, on address, and on phone
• Compare iff records are identical in at least one
• Misses similar records with small differences in all three fields
PLICIT
could also consider fancier LSH, e.g., for string edit distance

Getting more formal with LSH
• Survey: [Andoni & Indyk, CACM 2008]
• A family \( \mathcal{H} \) of hash functions mapping \( \mathbb{R}^d \) to some universe \( U \) is called \((R, cR, P_1, P_2)\)-sensitive if for any two points \( p, q \in \mathbb{R}^d \):
  • If \( \|p - q\| \leq R \) then \( \Pr_{h \in \mathcal{H}}[h(p)=h(q)] \geq P_1 \)
  • If \( \|p - q\| \geq cR \) then \( \Pr_{h \in \mathcal{H}}[h(p)=h(q)] \leq P_2 \)
• Obviously you want \( P_1 > P_2 \)
• Ongoing quest of finding LSH for various metrics
  • Hamming, \( L_1 \) (Manhattan), \( L_2 \) (Euclidean), \( L_p \)
  • Jaccard (\( \rightarrow \) Minhash), EMD, ...