Near-Neighbor Search

Applications
Matrix Formulation
Minhashing
Example Application: Face Recognition

- We have a database of (say) 1 million face images.
- We want to find the most similar images in the database.
- Represent faces by (relatively) invariant values, e.g., ratio of nose width to eye width.
Face Recognition – (2)

◆ Each image represented by a large number (say 1000) of numerical features.

◆ **Problem**: given a face, find those in the DB that are close in at least $\frac{3}{4}$ (say) of the features.
Face Recognition – (3)

- **Many-one problem**: given a new face, see if it is close to any of the 1 million old faces.
- **Many-Many problem**: which pairs of the 1 million faces are similar.
Simple Solution

- Represent each face by a vector of 1000 values and score the comparisons.
- Sort-of OK for many-one problem.
- Out of the question for the many-many problem \((10^6 \times 10^6 \times 1000/2\) numerical comparisons).
- We can do better!
Multidimensional Indexes Don’t Work

New face: [6, 14, ...]

Dimension 1 =

- 0-4
- 5-9
- 10-14
- ...

Maybe look here too, in case of a slight error.

Surely we’d better look here.

But the first dimension could be one of those that is not close. So we’d better look everywhere!
Another Problem: Entity Resolution

- Two sets of 1 million name-address-phone records.
- Some pairs, one from each set, represent the same person.
- Errors of many kinds:
  - Typos, missing middle initial, area-code changes, St./Street, Bob/Robert, etc., etc.
Entity Resolution – (2)

◆ Choose a scoring system for how close names are.
  ♦ Deduct so much for edit distance > 0; so much for missing middle initial, etc.

◆ Similarly score differences in addresses, phone numbers.

◆ Sufficiently high total score -> records represent the same entity.
Simple Solution

- Compare each pair of records, one from each set.
- Score the pair.
- Call them the same if the score is sufficiently high.
- Unfeasible for 1 million records.
- We can do better!
Example: Similar Customers

- **Common pattern**: looking for sets with a relatively large intersection.
- Represent a customer, e.g., of Netflix, by the set of movies they rented.
- Similar customers have a relatively large fraction of their choices in common.
Example: Similar Products

- Dual view of product-customer relationship.
- Products are similar if they are bought by many of the same customers.
- E.g., movies of the same genre are typically rented by similar sets of Netflix customers.
  - Tricky: Sony and Samsung TV’s are “similar,” but not typically bought by the same customers.
Yet Another Problem: Finding Similar Documents

Given a body of documents, e.g., the Web, find pairs of docs that have a lot of text in common, e.g.:
- Mirror sites, or approximate mirrors.
- Plagiarism, including large quotations.
- Repetitions of news articles at news sites.
Complexity of Document Similarity

- For the face problem, there is a way to represent a big image by a (relatively) small data-set.
- Entity records represent themselves.
- How do you represent a document so it is easy to compare with others?
Complexity – (2)

◆ Special cases are easy, e.g., identical documents, or one document contained verbatim in another.

◆ General case, where many small pieces of one doc appear out of order in another, is very hard.
Roadmap

- Similar customers
- Similar products
- Documents

Sets or Boolean matrices

Technique: Minhashing

Signatures

Technique: Locality-Sensitive Hashing

Buckets containing mostly similar items

- Technique: Shingling
- Face-recognition
- Entity-resolution
Representing Documents for Similarity Search

1. Represent doc by its set of *shingles* (or *k*-grams).

2. Summarize shingle set by a *signature* = small data-set with the property:
   - Similar documents are very likely to have “similar” signatures.
   - At that point, doc problem becomes finding similar sets.
Shingles

◆ A $k$-shingle (or $k$-gram) for a document is a sequence of $k$ characters that appears in the document.

◆ Example: $k=2$; $doc = abcab$. Set of 2-shingles = $\{ab, bc, ca\}$.
   - Option: regard shingles as a bag, and count ab twice.
Shingles: Compression Option

- To compress long shingles, we can hash them to (say) 4 bytes.
- Represent a doc by the set of hash values of its $k$-shingles.
- Two documents could (rarely) appear to have shingles in common, when in fact only the hash-values were shared.
MinHashing

Data as Sparse Matrices
Jaccard Similarity Measure
Constructing Signatures
Basic Data Model: Sets

◆ Many similarity problems can be couched as finding subsets of some universal set that have large intersection.

◆ **Examples** include:
  1. Documents represented by their set of shingles (or hashes of those shingles).
  2. Similar customers or products.
From Sets to Boolean Matrices

- **Rows** = elements of the universal set.
- **Columns** = sets.
- 1 in the row for element $e$ and the column for set $S$ iff $e$ is a member of $S$. 
### In Matrix Form

<table>
<thead>
<tr>
<th></th>
<th>S</th>
<th>T</th>
<th>U</th>
<th>V</th>
<th>W</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>c</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>d</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>e</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>f</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>g</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>h</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

$S = \{a, b, c, e, f\}$  
$T = \{a, d, f, g, h\}$  
$U = \{b, e\}$  
$V = \{a, b, c, f, g, h\}$  
$W = \{d, e, f, g\}$
Documents in Matrix Form

- **Rows** = shingles (or hashes of shingles).
- **Columns** = documents.
- 1 in row $r$, column $c$ iff document $c$ has shingle $r$.
- Expect the matrix to be sparse.
Aside

- We might not really represent the data by a boolean matrix.
- Sparse matrices are usually better represented by the list of places where there is a non-zero value.
  - E.g., movies rented by a customer, shingle-sets.
- But the matrix picture is conceptually useful.
Assumptions

1. Number of items allows a small amount of main-memory/item.
   - E.g., main memory = Number of items * 1000

2. Too many items to store anything in main-memory for each pair of items.
Similarity of Columns

- **Remember:** a column is the set of rows in which it has 1.

- The *similarity* of columns $C_1$ and $C_2 = Sim(C_1, C_2)$ is the ratio of the sizes of the intersection and union of $C_1$ and $C_2$.

  - $Sim(C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} = \text{Jaccard similarity.}$
**Example: Jaccard Similarity**

<table>
<thead>
<tr>
<th></th>
<th>C₁</th>
<th>C₂</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>*</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>**</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>**</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Sim \( (C₁, C₂) = \frac{2}{5} = 0.4 \)
Outline: Finding Similar Columns

1. Compute signatures of columns = small summaries of columns.
   - Read from disk to main memory.

2. Examine signatures in main memory to find similar signatures.
   - **Essential**: similarities of signatures and columns are related.

3. **Optional**: check that columns with similar signatures are really similar.
Warnings

1. Comparing all pairs of signatures may take too much time, even if not too much space.
   - A job for Locality-Sensitive Hashing.

2. These methods can produce false negatives, and even false positives if the optional check is not made.
Signatures

◆ Key idea: “hash” each column $C$ to a small signature $Sig(C)$, such that:

1. $Sig(C)$ is small enough that we can fit a signature in main memory for each column.

2. $Sim(C_1, C_2)$ is the same as the “similarity” of $Sig(C_1)$ and $Sig(C_2)$.
An Idea That Doesn’t Work

◆ Pick 100 rows at random, and let the signature of column $C$ be the 100 bits of $C$ in those rows.

◆ Because the matrix is sparse, many columns would have 00...0 as a signature, yet be very dissimilar because their 1’s are in different rows.
Four Types of Rows

Given columns $C_1$ and $C_2$, rows may be classified as:

<table>
<thead>
<tr>
<th></th>
<th>$C_1$</th>
<th>$C_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$b$</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>$c$</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>$d$</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Also, $a = \# \text{ rows of type } a$, etc.

Note $\text{Sim} (C_1, C_2) = \frac{a}{a + b + c}$. 
Minhashing

- Imagine the rows permuted randomly.
- Define “hash” function $h(C) =$ the number of the first (in the permuted order) row in which column $C$ has 1.
- Use several (100?) independent hash functions to create a signature.
Minhashing Example

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Input matrix

$\begin{array}{cccc}
1 & 0 & 1 & 0 \\
1 & 0 & 0 & 1 \\
0 & 1 & 0 & 1 \\
0 & 1 & 0 & 1 \\
1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 \\
\end{array}$

Signature matrix $M$

$\begin{array}{cccc}
2 & 1 & 2 & 1 \\
2 & 1 & 4 & 1 \\
1 & 2 & 1 & 2 \\
\end{array}$
Surprising Property

- The probability (over all permutations of the rows) that $h(C_1) = h(C_2)$ is the same as $Sim(C_1, C_2)$.
- Both are $a / (a + b + c)!$
- **Why?**
  - Look down columns $C_1$ and $C_2$ until we see a 1.
  - If it’s a type-$a$ row, then $h(C_1) = h(C_2)$. If a type-$b$ or type-$c$ row, then not.
Similarity for Signatures

The similarity of signatures is the fraction of the rows in which they agree.
- Remember, each row corresponds to a permutation or “hash function.”
Min Hashing – Example

Input matrix

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Signature matrix $M$

<table>
<thead>
<tr>
<th></th>
<th>2</th>
<th>1</th>
<th>2</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Similarities:

Col/Col | 1-3 | 2-4 | 1-2 | 3-4
---|------|------|------|------
   | 0.75 | 0.75 | 0   | 0   |
Sig/Sig | 0.67 | 1.00 | 0   | 0   |
Minhash Signatures

- Pick (say) 100 random permutations of the rows.
- Think of $\text{Sig}(C)$ as a column vector.
- Let $\text{Sig}(C)[i] = \text{according to the } i\text{th permutation, the number of the first row that has a 1 in column } C$. 
Implementation – (1)

- Suppose 1 billion rows.
- Hard to pick a random permutation from 1...billion.
- Representing a random permutation requires 1 billion entries.
- Accessing rows in permuted order leads to thrashing.
Implementation – (2)

- A good approximation to permuting rows: pick (say) 100 hash functions.
- For each column $c$ and each hash function $h_i$, keep a “slot” $M(i, c)$ for that minhash value.
 Implementation – (3)

\[
\text{for each row } r \\
\text{for each column } c \\
\text{if } c \text{ has 1 in row } r \\
\text{for each hash function } h_i \text{ do} \\
\text{if } h_i(r) \text{ is a smaller value than } M(i, c) \text{ then} \\
M(i, c) := h_i(r); \\
\]
Example

\[ h(x) = x \mod 5 \]
\[ g(x) = 2x + 1 \mod 5 \]

<table>
<thead>
<tr>
<th>Row</th>
<th>C1</th>
<th>C2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ h(1) = 1 \]
\[ g(1) = 3 \]
\[ h(2) = 2 \]
\[ g(2) = 0 \]
\[ h(3) = 3 \]
\[ g(3) = 2 \]
\[ h(4) = 4 \]
\[ g(4) = 4 \]
\[ h(5) = 0 \]
\[ g(5) = 1 \]

<table>
<thead>
<tr>
<th></th>
<th>Sig1</th>
<th>Sig2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>-</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>0</td>
</tr>
</tbody>
</table>
Implementation – (4)

- If data is stored row-by-row, then only one pass is needed.
- If data is stored column-by-column
  - E.g., data is a sequence of documents represent it by (row-column) pairs and sort once by row.
  - Saves cost of computing $h(r)$ many times.