Near-Neighbor Search

Applications
Matrix Formulation
Minhashing
Example Problem --- Face Recognition

- We have a database of (say) 1 million face images.
- We are given a new image and 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 the features of a new 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 ($10^6 \times 10^6 \times 1000$ 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!
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.
- Find mirror sites, approximate mirrors, plagiarism, quotation of one document in another, “good” document with random spam, etc.
Complexity of Document Similarity

- The face problem had a way of representing 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.
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 resembles the previous two problems.
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: Aside

◆ Although we shall not discuss it, shingles are a powerful tool for characterizing the topic of documents.
  ♦ $k = 5$ is the right number; $(\#\text{characters})^5 \gg \# \text{ shingles in typical document}.$

◆ Example: “ng av” and “ouchd” are most common in sports articles.
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
Roadmap

- Market baskets
  - Documents
  - Other apps

- Boolean matrices
  - Minhashing
    - Signatures
      - Locality-Sensitive Hashing
        - Entity-resolution
        - Face-recognition
        - Other apps

- Shingling
Boolean Matrix Representation

- Data in the form of subsets of a universal set can be represented by a (typically sparse) matrix.
- **Examples** include:
  1. Documents represented by their set of shingles (or hashes of those shingles).
Matrix Representation of Item/Basket Data

- **Columns** = items.
- **Rows** = baskets.
- Entry \((r, c)\) = 1 if item \(c\) is in basket \(r\); = 0 if not.
- Typically matrix is almost all 0’s.
In Matrix Form

<table>
<thead>
<tr>
<th></th>
<th>m</th>
<th>c</th>
<th>p</th>
<th>b</th>
<th>j</th>
</tr>
</thead>
<tbody>
<tr>
<td>{m,c,b}</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>{m,p,b}</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>{m,b}</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>{c,j}</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>{m,p,j}</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>{m,c,b,j}</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>{c,b,j}</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>{c,b}</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Documents in Matrix Form

- **Columns** = documents.
- **Rows** = shingles (or hashes of shingles).
- 1 in row $r$, column $c$ iff document $c$ has shingle $r$.
- Again 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., baskets, 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 * 100

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

Think of a column as the set of rows in which it has 1.

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

$\text{Sim} (C_1, C_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|} = \text{Jaccard measure}$. 
Example

\[
\begin{array}{ccc}
C_1 & C_2 \\
0 & 1 & * \\
1 & 0 & * \\
1 & 1 & ** \\
1 & 1 & ** \\
0 & 0 \\
1 & 1 & ** \\
0 & 1 & *
\end{array}
\]

\[
\text{Sim (} C_1, C_2 \text{)} = \frac{2}{5} = 0.4
\]
Outline of Algorithm

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

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

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Signature matrix $M$

<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

Similarities:

<table>
<thead>
<tr>
<th></th>
<th>1-3</th>
<th>2-4</th>
<th>1-2</th>
<th>3-4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Col/Col</td>
<td>0.75</td>
<td>0.75</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Sig/Sig</td>
<td>0.67</td>
<td>1.00</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
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)

- Number of rows = 1 billion (say).
- Hard to pick a random permutation from 1...billion.
- Representing a random permutation requires 1 billion entries.
- Accessing rows in permuted order is tough!
  - The number of passes would be prohibitive.
Implementation --- (2)

1. Pick (say) 100 hash functions.
2. For each column $c$ and each hash function $h_i$, keep a “slot” $M(i, c)$ for that minhash value.
Implementation --- (3)

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

\begin{itemize}
  \item Needs only one pass through the data.
\end{itemize}
Example

<table>
<thead>
<tr>
<th>Row</th>
<th>C1</th>
<th>C2</th>
<th>Sig1</th>
<th>Sig2</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

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