Approximate Computation of Object Distances by Locality-Sensitive Hashing

Selim Mimaroglu, Dan Simovici
Computer Science Department
University of Massachusetts at Boston

July 14, 2008
Content

- Problem Statement
- Background
- Solution
- Results and Conclusions
Problem Statement - 1

- A distance matrix contains pairwise distances between objects.

<table>
<thead>
<tr>
<th></th>
<th>obj&lt;sub&gt;1&lt;/sub&gt;</th>
<th>obj&lt;sub&gt;2&lt;/sub&gt;</th>
<th>obj&lt;sub&gt;3&lt;/sub&gt;</th>
<th>obj&lt;sub&gt;4&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj&lt;sub&gt;1&lt;/sub&gt;</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>obj&lt;sub&gt;2&lt;/sub&gt;</td>
<td>7</td>
<td>0</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>obj&lt;sub&gt;3&lt;/sub&gt;</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>obj&lt;sub&gt;4&lt;/sub&gt;</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>

- Many clustering algorithms operate on a distance matrix.

- Distance matrix of a database having \( N \) objects requires \( \frac{N^2 - N}{2} \) computations.
Problem Statement - 2

Compute distance matrix approximately, but faster.

<table>
<thead>
<tr>
<th></th>
<th>obj₁</th>
<th>obj₂</th>
<th>obj₃</th>
<th>obj₄</th>
</tr>
</thead>
<tbody>
<tr>
<td>obj₁</td>
<td>0</td>
<td>7</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>obj₂</td>
<td>7</td>
<td>0</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>obj₃</td>
<td>3</td>
<td>5</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>obj₄</td>
<td>2</td>
<td>4</td>
<td>6</td>
<td>0</td>
</tr>
</tbody>
</table>
Background - 1

- “Approximate Nearest Neighbors: Towards Removing the Curse of Dimensionality” by Indyk, and Motwani
- “Similarity Search in High Dimensions via Hashing” by Gionis, Indyk, and Motwani
- Works with high dimensional data
- Hash data points
- Probability of collision is higher for close objects
- Probability of collision is low for far apart objects
Background - 2

- Works well with binary data

\[ \mathcal{D} \]

<table>
<thead>
<tr>
<th>row</th>
<th>( i_1 )</th>
<th>( i_2 )</th>
<th>( i_3 )</th>
<th>( i_4 )</th>
<th>( i_5 )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

- Works well with the Hamming distance
- Each hash function (probe) randomly select attributes for projecting
Locality Sensitive Hashing

- Choose the attributes for projection randomly
- \( K = \{i_1, i_3, i_5\} \), \( h_K \) is binary equivalent of the projection.

\[
\begin{array}{c|ccccc}
 r & i_1 & i_2 & i_3 & i_4 & i_5 \\
\hline
 1 & 1 & 0 & 0 & 1 & 1 \\
 2 & 0 & 1 & 1 & 0 & 0 \\
 3 & 1 & 0 & 1 & 0 & 0 \\
 4 & 1 & 1 & 0 & 1 & 0 \\
 5 & 0 & 1 & 1 & 1 & 1 \\
 6 & 0 & 0 & 1 & 1 & 1 \\
 7 & 1 & 0 & 1 & 0 & 1 \\
 8 & 1 & 1 & 0 & 0 & 1 \\
 9 & 0 & 1 & 1 & 1 & 0 \\
\end{array}
\]

Binary Data

\[
\begin{array}{cccc}
000 & 001 & 010 & 011 \\
\{\} & \{\} & \{2,9\} & \{5,6\} \\
\{1,8\} & \{3\} & \{7\}
\end{array}
\]

Blocks Created by \( h_K \)
Approximate Computation of Object Distances - 1

- Family of random projections (LSH)
- Scales up with the number of objects
- Much faster than “brute-force”
- Very low memory requirements
- Very good approximation
Choose the attributes for projection randomly

\[ K = \{i_1, i_3, i_5\}, \ h_K \text{ is binary equivalent of the projection.} \]
Choose the attributes for projection randomly

\[ K = \{i_2, i_4, i_5\}, \ g_K \text{ is binary equivalent of the projection.} \]

\[ \begin{array}{c|ccccc}
 r & i_1 & i_2 & i_3 & i_4 & i_5 \\
 \hline
 1 & 1 & 0 & 0 & 1 & 1 \\
 2 & 0 & 1 & 1 & 0 & 0 \\
 3 & 1 & 0 & 1 & 0 & 0 \\
 4 & 1 & 1 & 0 & 1 & 0 \\
 5 & 0 & 1 & 1 & 1 & 1 \\
 6 & 0 & 0 & 1 & 1 & 1 \\
 7 & 1 & 0 & 1 & 0 & 1 \\
 8 & 1 & 1 & 0 & 0 & 1 \\
 9 & 0 & 1 & 1 & 1 & 0 \\
\end{array} \]

Binary Data

\begin{array}{cccc}
 000 & 001 & 010 & 011 \\
 \{3\} & \{7\} & \{} & \{1,6\} \\
 100 & 101 & 110 & 111 \\
 \{2\} & \{8\} & \{4,9\} & \{5\} \\
\end{array} \]

Blocks Created by \( g_K \)
Let $m$ be number of random hash functions (probes),
$n$ be the total number of attributes,
$k$ be projection size,
$d = d(u, v)$ is the Hamming distance.
Expected number of collisions between objects $u, v$ is

$$E(C(u, v)) \approx m \cdot \left(1 - \frac{d}{n - k}\right)^k$$

We estimate the distance between $u$ and $v$ as

$$d(u, v) \approx (n - k) \left(1 - \left(\frac{E(C(u, v))}{m}\right)^{\frac{1}{k}}\right) \quad (1)$$
Create $m$ hash functions, each projecting on randomly chosen $k$ attributes

Create $N \times N$ matrix, simultaneous occurrence matrix (SOM) which keeps track of collisions

Scan each block, increase the corresponding entry in SOM for each pair by 1.
  - For example, for a block having three elements $\{a, b, c\}$, the collision counts of the pairs: $(a, b), (a, c), (b, c)$ are increased by 1 in SOM.

Compute approximate distance matrix from SOM using (1)
Approximate Computation of Object Distances - 6

- Assume a 5-object dataset $D$ with $n = 4$ attributes, $k = 1$ projection size, and $m = 3$ probes, whose bit vectors are

$$
\begin{pmatrix}
0 \\
1 \\
1 \\
1 \\
0
\end{pmatrix}, \begin{pmatrix}
0 \\
1 \\
0 \\
1 \\
0
\end{pmatrix}, \text{ and } \begin{pmatrix}
1 \\
0 \\
1 \\
0 \\
0
\end{pmatrix}
$$

- For example, objects 2 and 4 occur in the same clusters 3 times.

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

SOM Approximate Distance Matrix
Scalability and Computation Time - 8
We calculated the cophenetic correlation coefficient between our approximate distance matrix $D$, and the Hamming distance matrix $H$.

The averages of the matrices $D$ and $H$ are denoted by $\bar{d}$ and $\bar{h}$, respectively.

Cophenetic correlation coefficient, $c$

\[
c = \frac{\sum (D_{ij} - \bar{d})(H_{ij} - \bar{h})}{\sqrt{\sum (D_{ij} - \bar{d})^2 \sum (H_{ij} - \bar{h})^2}}.\]

$c \in [0, 1]$, higher values indicate better correlation.
Results - 1

Cophasetic Correlation Coefficient
1000 data points, 20 attributes

# probes (m)

k=1
k=2
k=3
k=4
k=5
k=6
k=7
Results - 2 (High Dimensions)

Cophenetic Correlation Coefficient
1000 data points, 100 attributes
Results - 3 (Many Probes)

Cophenetic Correlation Coefficient
1000 data points, 20 attributes, k=2

#probes (m)
Parallel Computation

- Our algorithm is highly parallelizable.
- Each hash function (probe) is computed by a Java thread.
- Java threads are converted to operating system threads.
- Apple - Mac Pro having 8 cores, running on Mac OS X Leopard.

<table>
<thead>
<tr>
<th># probes (m)</th>
<th>Total Time (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2171</td>
</tr>
<tr>
<td>3</td>
<td>2045</td>
</tr>
<tr>
<td>4</td>
<td>2132</td>
</tr>
<tr>
<td>5</td>
<td>2269</td>
</tr>
<tr>
<td>6</td>
<td>2281</td>
</tr>
<tr>
<td>7</td>
<td>2442</td>
</tr>
<tr>
<td>8</td>
<td>2484</td>
</tr>
</tbody>
</table>

Figure: Parallel computation results on a database having 15,000 data points.
Approximate Computation of Object Distances by Locality-Sensitive Hashing

Selim Mimaroglu, Dan Simovici
Computer Science Department
University of Massachusetts at Boston
smimarog@cs.umb.edu