

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.

	<i>obj₁</i>	<i>obj₂</i>	<i>obj₃</i>	<i>obj₄</i>
<i>obj₁</i>	0	7	3	2
<i>obj₂</i>	7	0	5	4
<i>obj₃</i>	3	5	0	6
<i>obj₄</i>	2	4	6	0

- ▶ 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

	obj_1	obj_2	obj_3	obj_4
obj_1	0	7	3	2
obj_2	7	0	5	4
obj_3	3	5	0	6
obj_4	2	4	6	0

- ▶ Compute distance matrix approximately, but faster.

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}

<i>row</i>	i_1	i_2	i_3	i_4	i_5
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

- ▶ 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.

\mathcal{D}

r	i_1	i_2	i_3	i_4	i_5
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

Binary Data

000	001	010	011
{}	{}	{2,9}	{5,6}
100	101	110	111
{4}	{1,8}	{3}	{7}

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

Approximate Computation of Object Distances - 2

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

\mathcal{D}

r	i_1	i_2	i_3	i_4	i_5
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

Binary Data

000	001	010	011
{ }	{ }	{2,9}	{5,6}
100	101	110	111
{4}	{1,8}	{3}	{7}

Blocks Created by h_K

Approximate Computation of Object Distances - 3

- ▶ Choose the attributes for projection randomly
- ▶ $K = \{i_2, i_4, i_5\}$, g_K is binary equivalent of the projection.

\mathcal{D}

r	i_1	i_2	i_3	i_4	i_5
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

Binary Data

000	001	010	011
{3}	{7}	{}	{1,6}
100	101	110	111
{2}	{8}	{4,9}	{5}

Blocks Created by g_K

Approximate Computation of Object Distances - 4

- ▶ Let m be number of random hash functions (*probes*),
- ▶ n be the total number of attributes,
- ▶ k be projection size,
- ▶ $d = d(\mathbf{u}, \mathbf{v})$ is the Hamming distance.
- ▶ Expected number of collisions between objects \mathbf{u}, \mathbf{v} is

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

- ▶ We estimate the distance between \mathbf{u} and \mathbf{v} as

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

Approximate Computation of Object Distances - 5

- ▶ 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 \mathcal{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.

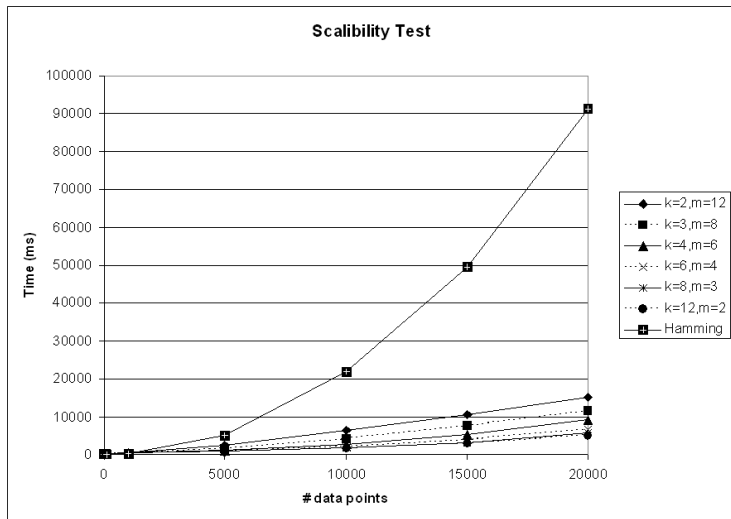
	1	2	3	4	5
1	3	0	2	0	2
2	0	3	1	3	1
3	2	1	3	1	1
4	0	3	1	3	1
5	2	1	1	1	3

SOM

	1	2	3	4	5
1	0	3	1	3	1
2	3	0	2	0	2
3	1	2	0	2	2
4	3	0	2	0	2
5	1	2	2	2	0

Approximate Distance Matrix

Scalability and Computation Time - 8



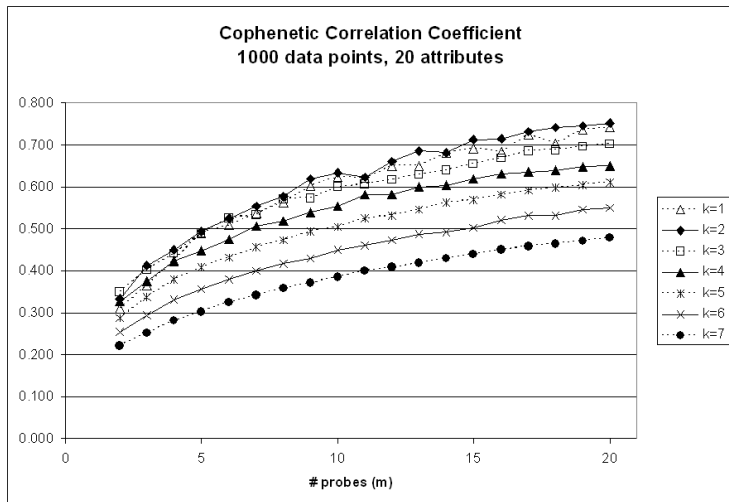
Cophenetic Correlation Coefficient

- ▶ 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

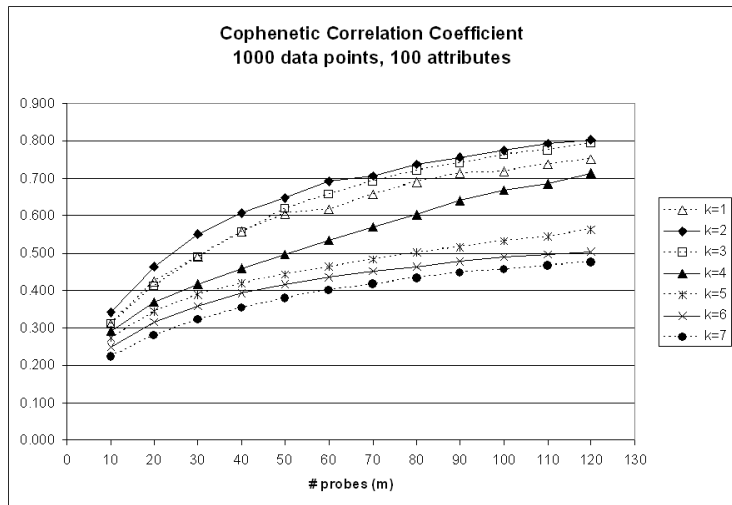
$$c = \frac{\sum (D_{ij} - \bar{d})(H_{ij} - \bar{h})}{\sqrt{\sum (D_{ij} - \bar{d})^2 \sum (H_{ij} - \bar{h})^2}}.$$

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

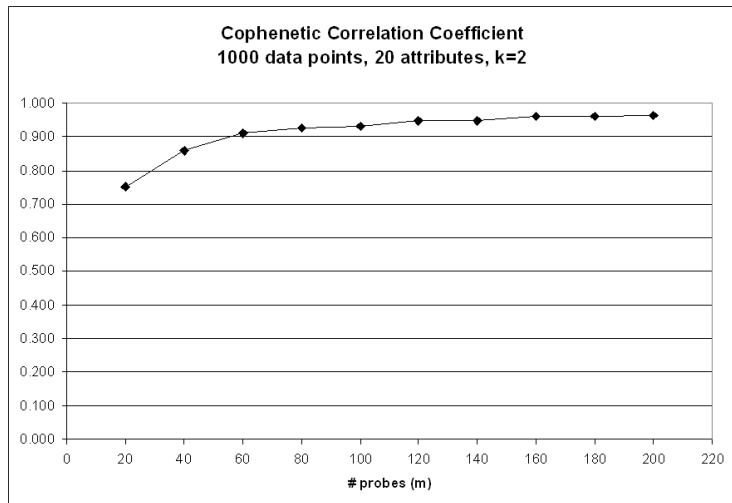
Results - 1



Results - 2 (High Dimensions)



Results - 3 (Many Probes)



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)

# probes (m)	Total Time (ms)
2	2171
3	2045
4	2132
5	2269
6	2281
7	2442
8	2484

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