Reciprocal Hash Tables for Nearest Neighbor Search

Xianglong Liu*, Junfeng He†,‡, Bo Lang

*Beihang University, †Columbia University, ‡Facebook

北京航空航天大學 BEIHANG UNIVERSITY

COLUMBIA UNIVERSITY IN THE CITY OF NEW YORK

AAAI-13 July 14-18, 2013 IAAI-13 July 16-18, 2013 BELLEVUE, WASHINGTON, USA

1. Background

Background

- **Nearest Neighbor Search** (NNS) is an important problem in domains like information retrieval, classification, and optimization
- **Locality-Sensitive Hashing** (LSH) gives the paradigm of hash based NNS, which achieves attractive performance than traditional tree based approaches
- **Various scenarios**: unsupervised, supervised, kernelized, multiple feature, multiple probes, multiple bits, etc.
- To improve the search performance, one successful technique is **building multiple hash tables** and returning points in multiple buckets

Main Issues

- Most of state-of-the-art hashing algorithms concentrate on how to generate hash functions for a single hash table
- Random selection as the most widely-used and general strategy faithfully improves the search performance, but usually needs a large number of hash tables

Motivation

- Similar to feature selection, we can select the most informative and independent hash functions, meanwhile considering the relations between tables
- Compatible with any hashing algorithm using different parameter settings or feature types.

2. Problem Formulation

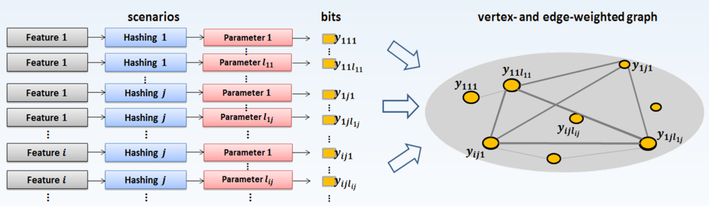
Notations

- A pool of B hash functions $H = \{h_1, h_2, \dots, h_B\}$ ($h_i: R^d \to \{-1,1\}$) with the index set $V = \{1,2, \dots, B\}$
- N training samples $X = [x_1, x_2, \dots, x_N] \in R^{d \times N}$, encoded by h_i into $Y_i = [h_i(x_1), h_i(x_2), \dots, h_i(x_N)]$

Definition

- The goal is to build L tables $\{T_l;\ l = 1, \dots, L\}$, each of which consists of K hash functions from H: $T_l = \{h_{l_1}, \dots, h_{l_K}; \{l_1, \dots, l_K\} \in V\}$.

Function Graph



Vertex and Edge Weighted Graph

$$G = (V, E, A, \pi)$$

- V is the vertex set corresponding to hash functions in H
- $\pi = [\pi_1, \dots, \pi_B]^T$ are the vertex weights
- $E \subset V \times V$ is the edge set
- $A = (a_{ij})$ are the edge weights

Selection Criteria

- Vertex weight: the accuracy of each hash function:

$$\pi_i = \exp(\gamma Y_i S Y_i^T)$$

- Edge weight: pairwise independence between functions

$$a_{ij} = \exp[-\lambda \sum_{y_i, y_j} p(y_i, y_j) \frac{p(y_i, y_j)}{p(y_i)p(y_j)}]$$

3. Reciprocal Hash Tables

Informative Tables

- Consist of the hash functions **preserving neighbor relationships and mutually independent: the dominant set** in the graph G

Theorem 1 *If $\mathbf{z}^*$ is a strict local solution of the program*

$$\begin{array}{ll} \max & \frac{1}{2}\mathbf{z}^T \hat{\mathbf{A}} \mathbf{z} \\ s.t. & \mathbf{z} \geq 0, \mathbf{1}^T \mathbf{z} = 1 \end{array}$$

Quadratic programming

where $\hat{\mathbf{A}} = \mathbf{\Pi A \Pi}$, and $\mathbf{\Pi} = diag(\pi)$, then its support $\sigma(\mathbf{z}^*) = \{i \in \mathcal{V} : z_i^* \neq 0\}$ is the normalized dominant set of graph $G = (\mathcal{V}, \mathcal{E}, \mathbf{A}, \pi)$.

- **Straightforward table construction strategy**: iteratively build hash tables by solving the above problems with respect to the remaining unselected hash functions in the pool H

Reciprocal Tables

- The redundancy among tables: tables should be complementary to each other, so that the nearest neighbors can be found in at least one of them.
- **Reciprocal table construction strategy**: for each table sequentially select the dominant hash functions that well separate the previous misclassified neighbors in a boosting manner
  - **Predict neighbor relations:** current l hash tables on the pair x_i and x_j $\quad \mathbf{p}_{ij} = \mathbf{d}_{ij}^l - \mu$; multi-table hamming distance $\mathbf{d}_{ij}^l = \min_{i=1,\dots,l} \sum_{h \in \mathcal{T}_i} \|h(\mathbf{x}_i) - h(\mathbf{x}_j)\|^2$
  - **Update the similarities:** weights on the misclassified neighbor pairs will be amplified to incur greater penalty $\quad \mathbf{S}_{ij} = \mathbf{S}_{ij}\omega_{ij}$; reweight scalar $\omega_{ij} = \begin{cases} \exp(-\alpha \mathbf{p}_{ij}), & \text{if } (i,j) \in \mathcal{M} \\ \exp(\alpha \mathbf{p}_{ij}), & \text{if } (i,j) \in \mathcal{C} \\ 0, & \text{otherwise} \end{cases}$

4. Experiments

Protocol

- Datasets: SIFT-1M: 1 Million 128-D SIFT and GIST-1M: 1 Million 960-D GIST
- Measure: Precision & recall using Hamming ranking and Hash table lookup

Results

- Over basic hashing algorithms

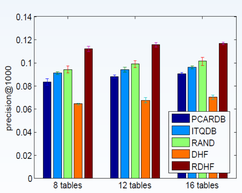
Table 2: PH2 (%) of multiple hashing tables over LSH, KLSH, and RMMH on GIST-1M and SIFT-1M.

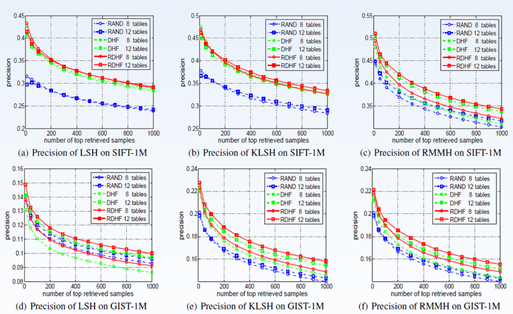
	Algorithms →	LSH			KLSH			RMMH		
	# Tables →	8	12	16	8	12	16	8	12	16
SIFT-1M	RAND	16.20	14.42	13.15	19.20	17.46	16.09	26.02	24.87	23.73
	DFH	26.88	23.24	16.22	**29.97**	25.34	18.86	**31.87**	30.62	28.63
	RDFH	**27.60**	**23.80**	**16.44**	29.40	**26.00**	**19.45**	31.23	**30.95**	**29.16**
GIST-1M	RAND	8.80	8.51	8.24	11.83	11.20	10.73	11.58	10.96	10.58
	DFH	9.21	9.30	8.72	**14.37**	13.63	11.88	13.28	12.68	11.59
	RDFH	**9.68**	**9.67**	**8.99**	14.27	**14.00**	**12.31**	**13.45**	**13.03**	**12.04**

- Over multiple hashing algorithms

Table 3: Performance (%) of hash lookup within Hamming radius 2 using multiple hashing tables on GIST-1M.

	Precision			Recall		
	8	12	16	8	12	16
PCARDB	8.42	8.41	8.31	4.90	5.79	6.74
ITQDB	8.93	8.72	8.60	5.24	6.50	7.57
RAND	9.34	9.26	9.10	5.82	7.01	8.19
DHF	7.36	7.13	6.98	1.11	2.12	3.37
RDHF	**11.32**	**10.82**	**10.44**	**6.89**	**9.40**	**10.88**





(a) Precision of LSH on SIFT-1M (b) Precision of KLSH on SIFT-1M (c) Precision of RMMH on SIFT-1M

(d) Precision of LSH on GIST-1M (e) Precision of KLSH on GIST-1M (f) Precision of RMMH on GIST-1M

Our proposed method significantly improves the performance of multiple hash tables

Conclusion

- a unified strategy for hash table construction, supporting different hashing algorithms and various scenarios
- a reciprocal strategy in a boosting manner to reduce the redundancy between hash tables