Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence (AAAI-17)

Boosting Complementary Hash Tables for Fast Nearest Neighbor Search

Xianglong Liu,[†] Cheng Deng,[‡] Yadong Mu,[§] Zhujin Li[†]
[†]State Key Lab of Software Development Environment, Beihang University, Beijing 100191, China
[‡]School of Electronic Engineering, Xidian University, Xi'an 710071, Shaanxi, China
[§]Institute of Computer Science and Technology, Peking University, Beijing 100080, China
xlliu@nlsde.buaa.edu.cn chdeng@mail.xidian.edu.cn myd@pku.edu.cn lizhujin@outlook.com

Abstract

Hashing has been proven a promising technique for fast nearest neighbor search over massive databases. In many practical tasks it usually builds multiple hash tables for a desired level of recall performance. However, existing multi-table hashing methods suffer from the heavy table redundancy, without strong table complementarity and effective hash code learning. To address the problem, this paper proposes a multi-table learning method which pursues a specified number of complementary and informative hash tables from a perspective of ensemble learning. By regarding each hash table as a neighbor prediction model, the multi-table search procedure boils down to a linear assembly of predictions stemming from multiple tables. Therefore, a sequential updating and learning framework is naturally established in a boosting mechanism, theoretically guaranteeing the table complementarity and algorithmic convergence. Furthermore, each boosting round pursues the discriminative hash functions for each table by a discrete optimization in the binary code space. Extensive experiments carried out on two popular tasks including Euclidean and semantic nearest neighbor search demonstrate that the proposed boosted complementary hash-tables method enjoys the strong table complementarity and significantly outperforms the state-of-the-arts.

Introduction

The past decades have witnessed the explosive growth of big data, especially visual data like images, videos, etc., which brings great challenges to scalable nearest neighbor search. Recently, hashing techniques have become one of the most promising solutions, owing to its attractive performance in a variety of applications including large-scale visual search (He et al. 2012; Song, Liu, and Meyer 2016), query-by-humming (Liu et al. 2016a), classification (Mu et al. 2014; Liu et al. 2016b) and recommendation (Liu et al. 2014b). As the pioneering work, Locality-Sensitive Hashing (LSH) first studied how to index similar data samples via representing them as adjacent hash codes (Datar et al. 2004), promising fast nearest neighbor search at the cost of sublinear time. The conventional LSH generates hash functions randomly and independently, and thus requires relatively long hash codes to meet the specific level of search quality. Instead of the random and data-independent way, a number of following hashing studies have been proposed to learn informative hash functions that can achieve satisfying performance using compact hash codes. These methods place great efforts in how to capture the neighboring relationships among the data, by exploiting different techniques including supervised learning (Xia et al. 2014; Lin, Shen, and van den Hengel 2015; Zhu et al. 2016; Kang, Li, and Zhou 2016), nonlinear mapping (Weiss, Torralba, and Fergus 2008; Liu et al. 2011; Liong et al. 2015; Li et al. 2016), discrete optimization (Gong et al. 2012; Liu et al. 2014a; Shen et al. 2015; Song, Liu, and Meyer 2016), structural information embedding (Yu et al. 2014; Wang, Si, and Shen 2015; Mu et al. 2016), multi-bit quantization (Kong and Li 2012; Mu et al. 2012; Deng et al. 2015; Li et al. 2016) etc.

Even encoding massive data into compact binary codes achieves compressed storage and efficient computations, it may be still beyond the requirement for balanced search performance in many practical tasks. To address this issues, in the literature LSH-based multiple table indexing is usually adopted to independently build a set of hash tables using LSH functions, which can faithfully improve the recall performance (Lv et al. 2007; Norouzi, Punjani, and Fleet 2012; Xia et al. 2013; Cheng et al. 2014). However, without eliminating the table redundancy it often requires a huge number of tables, at the cost of significantly sacrificing precision. To maximally cover the nearest neighbors using as few as possible tables, complementary multi-table methods have been studied to leverage the mutual benefits between tables. Xu et al. proposed a sequential learning method to build complementary hash tables, and obtained the promising performance with much fewer tables. Liu, He, and Lang studied a general multi-table construction strategy using bit selection over existing hashing algorithms. To further improve the search performance, Liu et al. introduced an exemplar-based feature fusion and reweighting strategy that can lead to discriminative and complementary hash tables accounting for input data with multiple views.

Although existing complementary multi-table methods can reach promising balanced search performance using a small number of tables, they still suffer from the table redundancy caused by both the heuristic updating scheme without a strong complementarity guarantee and the inefficient

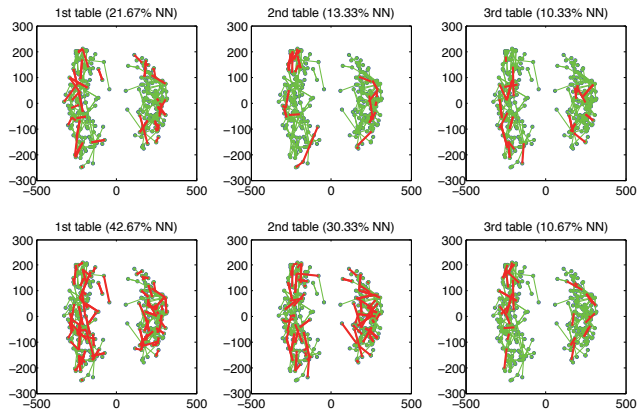


Figure 1: The nearest neighbor search results on a subset of SIFT-1M dataset: there are totally 300 samples (blue dots) projected in 2D space and 1-nearest neighbor for each sample (connected by green lines). The first and second row respectively demonstrates the results of the popular LSH method and our BCH method using three hash tables. The red lines indicate the correctly identified nearest neighbors (without repetition) using hash table lookup.

hash code learning procedure for each table. Motivated by the fact that multi-table based nearest neighbor search can be regarded as a combination of neighbor predictions stemming from multiple tables, we leverage the idea of ensemble learning and propose a boosted complementary hash tables (BCH) method which optimizes multiple tables and enjoys fast convergence. To the best of our knowledge, this is the first work that jointly formulates the multi-table learning problem for nearest neighbor search and meanwhile theoretically guarantees the table complementarity and algorithmic convergence. Our proposed method serves as a general framework directly applicable to many practical tasks like (He et al. 2012; Cheng et al. 2014) with a promising performance. Figure 1 illustrates the search results of the classical LSH tables and our BCH, where BCH retrieves much more nearest neighbors than LSH.

Boosted Complementary Hash Tables

Next, we introduce the notations and formulate our complementary hash-tables learning as a sequential neighbor prediction problem from the ensemble learning view.

Problem Formulation

Given a set of N training examples $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$, $\mathbf{x}_i \in \mathbb{R}^D$ of D dimension, our goal is to learn L complementary hash tables $\{\mathcal{H}_l\}_{l=1}^L$, each of which consists of B hash functions, *i.e.*, $\mathcal{H}_l = \{h_j^{(l)}(\cdot)\}_{j=1}^B$, where hash function $h_j^{(l)}(\cdot) : \mathbb{R}^D \to \{-1, 1\}$ is a binary mapping. In each table, using its B hash functions any point $\mathbf{x}_i$ can be encoded into a binary code $\mathbf{y}_i^{(l)} = [h_1^{(l)}(\mathbf{x}_i), \ldots, h_B^{(l)}(\mathbf{x}_i)]^{\mathrm{T}} \in \{-1, 1\}^B$, forming a code matrix $\mathbf{Y}^{(l)} = [\mathbf{y}_1^{(l)}, \ldots, \mathbf{y}_N^{(l)}]$ for all the N training data. For each table, we simply adopt the common linear projection based hash functions

$$h_j^{(l)}(\mathbf{x}) = \mathrm{sgn}(\mathbf{w}_j^{(l)\mathrm{T}} \mathbf{x}_i), \tag{1}$$

where $\mathbf{W}^{(l)} = [\mathbf{w}_1^{(l)}, \ldots, \mathbf{w}_B^{(l)}]$ are the projection vectors.

In practice, there are two types of neighbor sets for each sample $\mathbf{x}_i$, *i.e.*, homogenous neighbors $\mathcal{N}^o(\mathbf{x}_i)$ and heterogeneous neighbors $\mathcal{N}^e(\mathbf{x}_i)$. The former set refers to the k^o-nearest neighbors of $\mathbf{x}_i$ according to certain metric, and while the later contains the k^e neighbors far away from $\mathbf{x}_i$. The two types of neighbor sets can be represented in one similarity matrix $\mathbf{S} = (s_{ij})$: $s_{ij} = 1$, if $\mathbf{x}_j \in \mathcal{N}^o(\mathbf{x}_i)$; $s_{ij} = 0$, if $\mathbf{x}_j \in \mathcal{N}^e(\mathbf{x}_i)$; otherwise, $s_{ij} = 0$.

Intuitively, we expect that each sample shares similar hash codes with its homogeneous neighbors (*i.e.*, within a small Hamming distance to them), and different codes from its heterogenous neighbors. This means that a good hash table should index homogeneous neighbors in the adjacent buckets. When performing nearest neighbor search over multiple tables, the data samples belonging to the buckets with a quite small Hamming distance r_0 to the query are merged from each table and treated as the search results. This actually can be regarded as a simple combination of neighbor predictions generated by each hash table.

Formally, the neighbor prediction of the l-th table is determined according to the Hamming distance:

$$f^{(l)}(\mathbf{x}_i, \mathbf{x}_j) = \mathrm{sgn}(r_0 - d_H(\mathbf{y}_i^{(l)}, \mathbf{y}_j^{(l)})), \tag{2}$$

where $d_H(\mathbf{y}_i^{(l)}, \mathbf{y}_j^{(l)}) = \frac{1}{4}\|\mathbf{y}_i^{(l)} - \mathbf{y}_j^{(l)}\|^2$ is the Hamming distance, and r_0 is the specified Hamming distance threshold. $f^{(l)}(\mathbf{x}_i, \mathbf{x}_j) = 1$ indicates $\mathbf{x}_i$ and $\mathbf{x}_j$ are predicted as the homogenous neighbors, and otherwise heterogeneous ones.

Then totally using L hash tables, the neighbor prediction can be formulated as a joint one based on the linear combination of L result sets from all tables:

$$F^{(L)}(\mathbf{x}_i, \mathbf{x}_j) = \sum_{l=1}^{L} \beta^{(l)} f^{(l)}(\mathbf{x}_i, \mathbf{x}_j), \tag{3}$$

where $\beta^{(l)}$ is the weight for the l-th table.

With the neighbor prediction, we can quantitatively measure its quality according to the pre-defined similarity matrix. We introduce the exponential loss function of L hash tables over each neighbor pair $\mathbf{x}_i$ and $\mathbf{x}_j$:

$$c(\mathbf{x}_i, \mathbf{x}_j) = \begin{cases} e^{-s_{ij} F^{(L)}(\mathbf{x}_i, \mathbf{x}_j)} & , \quad |s_{ij}| = 1 \\ 0 & , \quad \text{otherwise} \end{cases} \tag{4}$$

The total cost over the training data should be minimized when pursuing the desired L complementary tables. Namely, the multi-table learning can be formulate as follows:

$$\min_{\{\mathbf{W}^{(l)}\}, \{\beta^{(l)}\}} \mathcal{J} = \sum_{ij} c(\mathbf{x}_i, \mathbf{x}_j) = \sum_{s_{ij} \neq 0} e^{-s_{ij} F^{(L)}(\mathbf{x}_i, \mathbf{x}_j)}, \tag{5}$$

with respect to hash function parameters $\{\mathbf{W}^{(l)}\}$ and the additive coefficients $\{\beta^{(l)}\}$ for L tables.

Boosting-based Sequential Learning

The multi-table learning has been formulated as a joint neighbor prediction problem. However, it is quite difficult to jointly find the optimal $\{\mathbf{W}^{(l)}\}$ and $\{\beta^{(l)}\}$ at the same time by directly solving the problem (5). Following the spirit of the well-known AdaBoost algorithm (Friedman, Hastie, and Tibshirani 1998), we can sequentially learn the hash tables $\mathcal{H}_l$, $l = 1 \dots L$, which together generate a reasonable prediction that minimizes the overall loss.

According to Equation (3), the function $F(\mathbf{x}_i, \mathbf{x}_j)$ is additive, and thus for l hash tables it can be decomposed:

$$F^{(l)}(\mathbf{x}_i, \mathbf{x}_j) = F^{(l-1)}(\mathbf{x}_i, \mathbf{x}_j) + \beta^{(l)} f^{(l)}(\mathbf{x}_i, \mathbf{x}_j). \quad (6)$$

The loss function in Equation (5) can be therefore simplified:

$$e^{-s_{ij}F^{(l)}(\mathbf{x}_i,\mathbf{x}_j)} = \pi_{ij}^{(l-1)} \cdot e^{-s_{ij}\beta^{(l)}f^{(l)}(\mathbf{x}_i,\mathbf{x}_j)}, \quad (7)$$

where the variable $\pi^{(l)} = \left(\pi_{ij}^{(l)}\right) \in \mathbb{R}^{N\times N}$, whose elements $\pi_{ij}^{(l)} = e^{-s_{ij}F^{(l)}(\mathbf{x}_i,\mathbf{x}_j)}$ maintain a probabilistic distribution over all neighbor pairs after normalization:

$$\pi_{ij}^{(l)} = \frac{e^{-s_{ij}F^{(l)}(\mathbf{x}_i,\mathbf{x}_j)}}{\sum_{s_{i'j'}\neq 0} e^{-s_{i'j'}F^{(l)}(\mathbf{x}_{i'},\mathbf{x}_{j'})}}. \quad (8)$$

By abandoning the high-order components in Taylor expansion and using the fact that both s_{ij} and function $f^{(l)}(\mathbf{x}_i, \mathbf{x}_j)$ are discrete over $\{-1, 1\}$, we can further approximate the second term in Equation (7) by:

$$e^{-s_{ij}\beta^{(l)}f^{(l)}(\mathbf{x}_i,\mathbf{x}_j)} \approx -s_{ij}\beta^{(l)}f^{(l)}(\mathbf{x}_i, \mathbf{x}_j) + const. \quad (9)$$

Subsequently, the loss function in the l-th round with $(l-1)$ tables learnt in previous rounds turns to

$$\mathcal{J} = -\sum_{s_{ij}\neq 0} \pi_{ij}^{(l-1)} \cdot s_{ij} \cdot f^{(l)}(\mathbf{x}_i, \mathbf{x}_j), \quad (10)$$

and Problem (5) can be reformulated equivalently as follows

$$\begin{aligned} \max_{\mathbf{W}^{(l)},\beta^{(l)}} \quad & \textstyle\sum_{z_{ij}^{(l)}\neq 0} z_{ij}^{(l)} \cdot f^{(l)}(\mathbf{x}_i, \mathbf{x}_j) \quad (11) \\ s.t. \quad & \mathbf{Y}^{(l)} = \text{sgn}(\mathbf{W}^{(l)\mathrm{T}}\mathbf{X}) \end{aligned}$$

where $z_{ij}^{(l)} = \pi_{ij}^{(l-1)} \cdot s_{ij}$, which is composition of the similarity $\mathbf{S}$ and the weight distribution $\pi^{(l-1)}$ in each round.

The above formulation indicates that we can learn L tables in a sequential manner, where for each table we can learn B hash functions by solving a similar problem like (11). Besides, since $\mathbf{Z}^{(l)} = (z_{ij}^{(l)})$ takes account of both the neighbor prediction on each pair and the overall prediction error of previous tables, it will help pursue the l-th table complementary to the previous $(l-1)$ tables, by amplifying the weights on the misclassified neighbor pairs of previous tables and discarding those on the correctly classified ones.

Let define the neighbor prediction error of the l-th table $\varepsilon^{(l)} = \sum_{s_{ij}f^{(l)}(\mathbf{x}_i,\mathbf{x}_j)<0} \pi_{ij}^{(l-1)}$. Then the optimal $\beta^{(l)}$ minimizing loss in Equation (10) can be obtained

$$\beta^{(l)} = \frac{1}{2}\ln\frac{1-\varepsilon^{(l)}}{\varepsilon^{(l)}}. \quad (12)$$

We can further guarantee that the learnt multiple tables together can monotonically decrease the prediction error:

Theorem 1 *The neighbor prediction error using L hash tables over the training set $\mathbf{X}$ is bounded above by:*

$$\mathcal{E} = \frac{1}{\|\mathbf{S}\|_0}\sum_{s_{ij}\neq 0} \mathbb{I}\left[s_{ij}F^{(L)}(\mathbf{x}_i, \mathbf{x}_j) \leq 0\right] \leq e^{-2\gamma^2 L}$$

where $\mathbb{I}[\cdot]$ is an indicator function, $\|\mathbf{S}\|_0$ denotes the number of the non-zero elements in $\mathbf{S}$, and $\gamma = \arg\min_{l=1,\dots,L}\gamma^{(l)}$ with $\gamma^{(l)} = \frac{1}{2} - \varepsilon^{(l)}$.

Discrete Hash Table Optimization

Now we have turn the multi-table construction into sequential learning in a boosting manner, where each round learns a hash table that minimizes the loss function in Problem (11).

For the l-th tables, Problem (11) can be approximated by relaxing the sign function to its real-valued surrogate:

$$\begin{aligned} \max_{\mathbf{W}^{(l)}} \quad & \textstyle\sum_{z_{ij}^{(l)}\neq 0} z_{ij}^{(l)} \cdot \mathbf{y}_i^{(l)\mathrm{T}}\mathbf{y}_j^{(l)} = \text{Tr}(\mathbf{Y}^{(l)\mathrm{T}}\mathbf{Z}^{(l)}\mathbf{Y}^{(l)}) \\ s.t. \quad & \mathbf{Y}^{(l)} = \text{sgn}(\mathbf{W}^{(l)\mathrm{T}}\mathbf{X}). \quad (13) \end{aligned}$$

where the optimal $\mathbf{W}^{(l)}$ is expected to maximally preserve the neighbor relations defined by $\mathbf{Z}^{(l)}$, mainly focusing on the misclassified neighbor pairs of the previous $(l-1)$ tables.

Without loss of the generality, for simplicity we omit the upper script l of all variables in the following discussions. Therefore, if we introduce and penalize the quantization loss $\|\mathbf{W}^{\mathrm{T}}\mathbf{X} - \mathbf{Y}\|_F$ on the binary codes $\mathbf{Y}$, we can rewrite the problem in (13) with a positive parameter $\lambda > 0$:

$$\begin{aligned} \max_{\mathbf{W},\mathbf{Y}} \quad & \text{Tr}\left(\mathbf{Y}\mathbf{Z}\mathbf{Y}^{\mathrm{T}}\right) - \lambda\|\mathbf{W}^{\mathrm{T}}\mathbf{X} - \mathbf{Y}\|_F \\ s.t. \quad & \mathbf{Y} \in \{-1, 1\}^{B\times N} \quad (14) \end{aligned}$$

To further guarantee the independence and balance of the learnt hash codes, we require the following two constraints:

$$\mathbf{W}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{W} = N\mathbf{I}_B \text{ and } \mathbf{W}^{\mathrm{T}}\mathbf{X}\mathbf{1} = \mathbf{0}. \quad (15)$$

The later one can be easily satisfied by centering the training data $\mathbf{X}$. Besides, $\text{Tr}(\mathbf{W}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{W}) = \text{Tr}(\mathbf{Y}\mathbf{Y}^{\mathrm{T}}) = BN$ holds. Therefore, the above problem can be simplified as

$$\begin{aligned} \max_{\mathbf{W},\mathbf{Y}} \quad & \text{Tr}\left(\mathbf{Y}\mathbf{Z}\mathbf{Y}^{\mathrm{T}} + \lambda\mathbf{W}^{\mathrm{T}}\mathbf{X}\mathbf{Y}^{\mathrm{T}}\right) \quad (16) \\ s.t. \quad & \mathbf{Y} \in \{-1, 1\}^{B\times N},\ \mathbf{W}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{W} = N\mathbf{I}_B. \end{aligned}$$

Alternating Optimization

The above hashing problem is essentially a nonlinear mixed-integer program involving a discrete variable $\mathbf{Y}$ and continuous one $\mathbf{W}$, which is generally difficult to solve or approximate. To this end, we propose a similar efficient alternating optimization algorithm to pursue the near-optimal solution, which consists of the following two subproblems with respect to $\mathbf{Y}$ ($\mathbf{Y}$-subproblem) and $\mathbf{W}$ ($\mathbf{W}$-subproblem) respectively. The algorithm improves and extends the optimizing technique in (Liu et al. 2014a) to a more general problem with more complex constraints in Problem (16).

$\mathbf{Y}$-subproblem: With $\mathbf{W}$ fixed, the problem turns to

$$\begin{aligned} \max_{\mathbf{Y}} \quad & \text{Tr}\left(\mathbf{Y}\mathbf{Z}\mathbf{Y}^{\mathrm{T}} + \lambda\mathbf{W}^{\mathrm{T}}\mathbf{X}\mathbf{Y}^{\mathrm{T}}\right) \\ s.t. \quad & \mathbf{Y} \in \{-1, 1\}^{B\times N} \quad (17) \end{aligned}$$

The above subproblem is NP-hard, due to the discrete variable $\mathbf{Y}$. However, we can find a near-optimal one using the powerful gradient method. Specifically, this can be solved using the signed gradient ascend method proposed in (Liu et al. 2014a), where in each iteration the hash codes can be updated in the following way

$$\mathbf{Y} := \text{sgn}\left(\mathcal{C}(2\mathbf{YZ} + \lambda\mathbf{W}^{\mathrm{T}}\mathbf{X}, \mathbf{Y})\right) \tag{18}$$

with the element-wise operator $\mathcal{C}(x, y) = x^{\mathbb{I}[x\neq 0]}y^{\mathbb{I}[x=0]}$.

According to signed gradient ascend method, using the iterative updating of hash codes, we can guarantee that the objective monotonically increases and the algorithm converges to a local optimal solution. This is can be easily verified following (Liu et al. 2014a). Therefore, in practice only a very few iterations are required to get the near-optimal hash codes $\mathbf{Y}^*$ using the above updating.

$\mathbf{W}$-subproblem: Next with $\mathbf{Y}$ fixed, the problem turns to

$$\begin{aligned} \max_{\mathbf{W}} \quad & \text{Tr}\left(\mathbf{W}^{\mathrm{T}}\mathbf{X}\mathbf{Y}^{\mathrm{T}}\right) \\ s.t. \quad & \mathbf{W}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{W} = N\mathbf{I}_B. \end{aligned} \tag{19}$$

As to this subproblem, the following theoretical result tells us that we can get the optimum $\mathbf{W}^*$ in analytical form by singular value decomposition (SVD):

Theorem 2 *Let* $\mathbf{M} = (\mathbf{X}\mathbf{X}^{\mathrm{T}})^{-\frac{1}{2}}$, $\mathbf{z} = \mathbf{M}\mathbf{X}\mathbf{1}$ *and* $\mathbf{J} = \mathbf{I}_N - \frac{1}{\|\mathbf{z}\|}\mathbf{z}\mathbf{z}^{\mathrm{T}}$. *Then with the SVD decomposition SVD* $\left(\mathbf{J}\mathbf{M}\mathbf{X}\mathbf{Y}^{\mathrm{T}}\right) = \mathbf{U}\mathbf{\Sigma}\mathbf{V}$, *the optimal solution to* $\mathbf{W}$*-subproblem will be*

$$\mathbf{W}^* = \sqrt{N}\mathbf{M}\left[\mathbf{U}\ \bar{\mathbf{U}}\right]\left[\mathbf{V}\ \bar{\mathbf{V}}\right]^{\mathrm{T}}, \tag{20}$$

where $\bar{\mathbf{U}}$ *and* $\bar{\mathbf{V}}$ *are the complement of* $\mathbf{U}$ *and* $\mathbf{V}$.

By iteratively alternating the two subproblems we can get the near-optimal hash code $\mathbf{Y}^*$ and projection vectors $\mathbf{W}^*$ for each table. Moreover, our theoretical results state that the alternating optimization will monotonically increase the objective, which guarantees the fast optimization convergence. See the experimental results in the supplementary material.

As to the initialization of $\mathbf{W}^{(l)}$ in each round, the popular eigen-decomposition solution like (Xu et al. 2011) can be adopted. Algorithm 1 lists the main steps of our boosted complementary hash-tables method.

Algorithm 1 Boosted Complementary Hash-Tables (BCH).

Input: the training data $\mathbf{X}$, the code length B of each table, the desired table number L.
Output: hash tables $\mathcal{H}_l = \{h_j^{(l)}\}_{j=1}^{B}$, $l = 1, \ldots, L$.
Initialize: the similarity matrix $\mathbf{S}$, the probabilistic distribution $\pi^{(0)} = \frac{1}{\|\mathbf{S}\|_0}|\mathbf{S}|$.
for $l = 1, \ldots, L$ **do**
  update $\mathbf{Z}^{(l)} = \pi^{(l-1)} \circ \mathbf{S}$;
  initialize $\mathbf{W}^{(l)}$ using eigen-decomposition;
  repeat
    $\mathbf{Y}$-subproblem: repeat updating hash codes $\mathbf{Y}^{(l)}$ according to (18), until converge;
    $\mathbf{W}$-subproblem: update projection vectors $\mathbf{W}^{(l)}$ according to (20);
  until converge
  update neighbor prediction $F^{(l)}$ according to (6) ;
  update the distribution $\pi^{(l)}$ according to (8) ;
end for

Experiments

In this section we will evaluate the proposed boosted complementary hash-tables (BCH for short) method. In the literature most of research devoted great efforts to pursuing informative hash codes for Hamming distance ranking or single hash table lookup. There are very few related work regarding multiple complementary hash tables from the indexing aspect, except the three existing methods: bit selection (BS) (Liu, He, and Lang 2013), complementary hashing (CH) (Xu et al. 2011), and multi-view complementary hash tables (MVCH) (Liu et al. 2015). All these methods concentrate on the table complementarity to pursue balanced retrieval performance in a sequential way. However, only our BCH jointly considers the complementarity among all tables and theoretically guarantees the convergence.

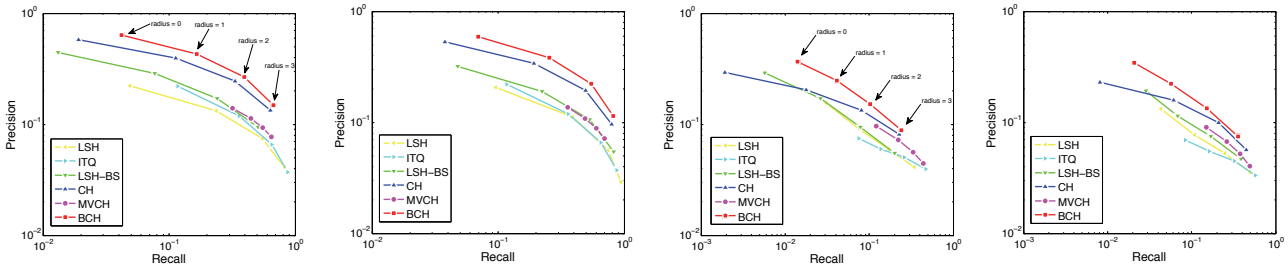
We also compare BCH to the state-of-the-art well-known unsupervised hashing algorithms: *Local Sensitive Hashing* (LSH) (Datar et al. 2004) and *Iterative Quantization* (ITQ) (Gong and Lazebnik 2011). In this case, when building L tables with B hash functions in each table, we adopt the common way that totally BL hash functions are generated first using each hashing algorithm, and then divided equally to L parts correspondingly forming L hash tables with B hash functions. This is similar to Multi-Index Hashing (Norouzi, Punjani, and Fleet 2012) that can speedup the search based on Hamming distance ranking.

Evaluation Protocols

In our experiments we respectively build a different number (*i.e.*, L) of hash tables using different methods. As to the parameter B, which has direct effects on the collision probability in nearest neighbor search, we fix it to the empirical value $\log_2 N$ for N database points (Norouzi, Punjani, and Fleet 2012; Slaney, Lifshits, and He 2012).

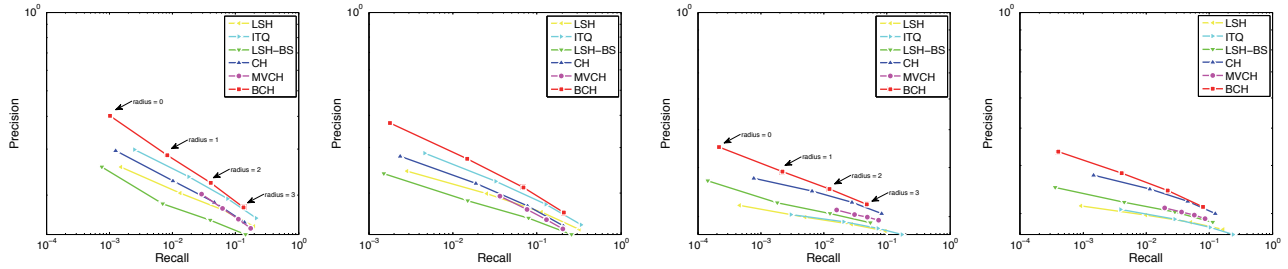
As we mainly focus on multi-table indexing technique, we adopt the common hashing search scheme: hash table lookup to evaluate the performance. In each hash table, those points falling within certain Hamming radius from the query code are returned and merged as the retrieved results. For efficiency problem, usually a small search radius (less than 4, and $r_0 = 4$) is used to avoid the expensive computation stemming from too many buckets with combinatorial explosion in each table (Liu et al. 2011).

In all experiments, for MVCH method we employ 300 exemplars generated by k-means clustering, and 5 nearest ones for the feature transformation of each sample. As to BCH, for each training sample we choose 50 homogenous neighbors and 100 heterogenous neighbors based on Euclidean distance. Moreover, the parameter λ is simply set to 1.0. All experiments are conducted on a workstation with Intel Xeon CPU E5-4607@2.60GHz and 48GB memory, and the results are averaged over 10 runs.



(a) 8 tables @ SIFT-1M (b) 16 tables @ SIFT-1M (c) 8 tables @ GIST-1M (d) 16 tables @ GIST-1M

Figure 2: Precision-Recall performance of different multi-table methods on SIFT-1M and GIST-1M



(a) 8 tables @ CIFAR-10 (b) 16 tables @ CIFAR-10 (c) 8 tables @ NUS-WIDE (d) 16 tables @ NUS-WIDE

Figure 3: Precision-Recall performance of different multi-table methods in image retrieval task on CIFAR-10 and NUS-WIDE

Euclidean Nearest Neighbor Search

Multi-table indexing has been widely used in the popular tasks including Euclidean and semantic nearest neighbor search. We first conduct experiments on the Euclidean one. We employ two widely-used large data sets: **SIFT-1M** and **GIST-1M**[1], consisting of one million 128-D SIFT and 960-D GIST features respectively. On both datasets, the groundtruth for each query is defined as the top ‰5 nearest neighbors with the smallest Euclidean distances. For each dataset, we construct a training and a testing set respectively with 10,000 and 3,000 random samples.

Figure 2(a) and (b) depict the precision-recall curves of different methods using 8 and 16 hash tables on SIFT-1M. Here, each table consists of hash codes with $B = 20$ bits. The performance curves are obtained by varying the lookup radius from 0 to 3. From the results, we can see that as we enlarge the search range using a large radius, more samples falling in the nearest buckets will be returned and thus the recall performance increases. However, the precision drops due to the false positive and redundant samples from multiple tables. The multi-table methods (BS, CH, MVCH and BCH) can alleviate the effects by considering the table complementarity, and thus outperform the basic hashing algorithms LSH and ITQ. Moreover, in all cases our BCH achieves the best performance with largest areas under the curves. This indicates that BCH can boost the table complementarity to cover more nearest neighbors using fewer tables than state-of-the-art multi-table methods.

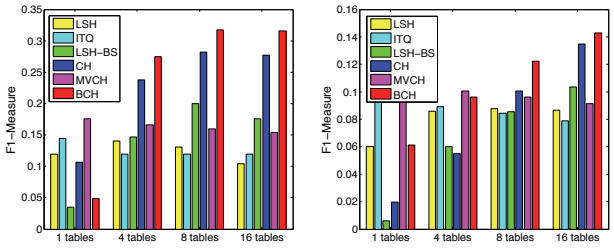
Figure 4(a) further investigates the overall performance with respect to the table number. We adopt the common metric F1-measure within Hamming radius 2 for comprehensive evaluation, which takes both precision and recall into consideration. It can be observed that most methods increase their search performances when using more hash tables, except LSH and ITQ. This further confirms the fact that the traditional hashing algorithms are not suitable for multi-table indexing, and while multi-table methods can achieve better performance. Compared to other multi-table methods, though BCH gives a low F1 performance when using 1 table, but it gains significant performance improvements when using more tables, and gets the best performance using 4 or more tables. This fact indicates that BCH enjoys the strongest table complementarity so that only a few tables are required to get a specified performance level, *e.g.*, BCH using 4 tables can get close F1-measure performance to CH using 8 tables. Similar observation can be obtained in the demo case in Figure 1.

Besides the results on SIFT-1M, we also compare different multi-table methods on GIST-1M, which contains GIST features of much higher dimension. With respect to different table numbers, Figure 2 (c) and (d) show the precision-recall curves, and Figure 4(b) depicts F1-measure bars. On GIST-1M, we can obtain same conclusions that (1) the complementary multi-table methods owns greater superiority than the conventional hashing algorithms. For instance, BS, CH and BCH significantly boosts the F1-measure when using multiple hash tables. (2) in most cases, BCH attains the best performance in term of both precision-recall curves (especially when the lookup radius less than 2) and F1-measure (up to *e.g.*, 23.8% gains compared to the best competitor CH

[1]http://corpus-texmex.irisa.fr

Table 1: Precision (%) and time cost (seconds) of different multi-table methods on four datasets.

METHOD	SIFT-1M			GIST-1M			CIFAR-10			NUS-WIDE		
	$L = 4$	$L = 16$	TRAIN TIME	$L = 4$	$L = 16$	TRAIN TIME	$L = 4$	$L = 16$	TRAIN TIME	$L = 4$	$L = 16$	TRAIN TIME
LSH	20.06	20.45	1.63	7.98	8.46	1.46	19.26	19.82	1.39	30.94	31.24	1.71
ITQ	22.22	-	-	12.19	9.27	79.65	22.13	21.26	17.81	34.65	32.24	68.10
LSH-BS	22.68	25.65	552.49	6.81	9.51	504.47	17.63	19.20	489.51	32.11	32.38	476.80
CH	38.45	40.03	53.20	14.59	16.03	91.04	22.72	23.45	49.69	36.78	37.43	128.91
MVCH	17.75	15.73	1106.85	11.62	10.58	1570.12	21.84	21.41	1214.07	34.69	34.40	1394.10
BCH	**41.30**	**46.17**	153.52	**15.85**	**17.15**	236.46	**25.03**	**26.13**	156.01	**38.63**	**39.10**	192.63



(a) 1-16 tables @ SIFT-1M (b) 1-16 tables @ GIST-1M

Figure 4: F1-Measure performance using different number of hash tables on SIFT-1M and GIST-1M

when using 8 hash tables).

Semantic Nearest Neighbor Search

Besides the Euclidean nearest neighbor search that can be applied to accelerating tasks like image matching, clustering etc., semantic neighbor search also has extensive applications in visual search, recommendation etc. Thus, we next evaluate BCH on semantic neighbor search for the image retrieval task. We employ two widely-used large image datasets: **CIFAR-10**[2] and **NUS-WIDE**[3]. CIFAR-10 contains 60K 32×32 color images of 10 classes and 6K images in each class, represented by a 384-D GIST feature. NUS-WIDE comprises over 269K images with 81 ground truth concept tags. We consider 25 most frequent tags and concatenate three presentative features including 128-D wavelet texture, 225-D block-wise color moments and 500-D SIFT-based BoW histograms as a 853-D feature for each image. Here, the groundtruth for each query is defined as those samples with common tags as the query.

Figure 3(a) and (b) report the precision-recall curves using 8 and 16 hash tables on CIFAR-10. We set the code length $B = 16$ for each table according to the database size. From the figures, we can get a similar conclusion that either building multiple hash tables or enlarging the lookup radius can find more nearest neighbors, subsequently improving the recall performance, but at the cost of precision drops. Fortunately, the complementary multi-table methods can balance precision and recall well using fewer tables. Here although the basic ITQ method performs better than LSH and BS, and very close to CH and MVCH, however, it neglects the reciprocal relations among the multiple tables, and thus gets a much lower performance than BCH.

In Figure 3(c) and (d) we further evaluate different methods over the much larger and more complex dataset NUS-WIDE (we use $B = 18$ bits for each table). Different from CIFAR-10, due to the complex semantics, ITQ on NUS-WIDE cannot well discover the neighbor relations using the basic image features. However, the complementary multi-table methods including BS, CH, MVCH and BCH, attempting to eliminate the redundancy between tables, can get much better performance when using different number of tables. Among them, our BCH still enjoys significant performance gains over all the baselines, owing to the adaptive aggregating of the learnt informative hash tables.

Hamming Distance Ranking over Multi-Tables

Besides table lookup, we can also adopt Hamming distance to rank the samples indexed in the tables. Specifically, the distance between the query $\mathbf{x}_q$ and any database sample $\mathbf{x}_i$ falling within the specified Hamming radius is computed as follows (Xu et al. 2011; Liu, He, and Lang 2013):

$$d(\mathbf{x}_q, \mathbf{x}_i) = \min_{l=1,\dots,L} d_H(\mathbf{y}_q^{(l)}, \mathbf{y}_i^{(l)}).$$

Such a distance can reflect how close it is between the database point and query. Based on this distance, all the candidates looked up from multiple tables can be ranked, incorporating the discriminative power of each table.

Table 1 investigates the average precision (AP) of the top 1,000 ranked results on SIFT-1M and GIST-1M. We can easily observe that our BCH can boost its precision when using more hash tables, while the others increase a little or even decrease. Moreover, in all cases BCH consistently achieves the best performance, *e.g.*, on SIFT-1M it respectively gets 7.4% and 15.3% precision gains over the best competitor CH when using 4 and 16 tables. On CIFAR-10 and NUS-WIDE, the reported AP performance of the top 500 ranked results further verifies our conclusion that with more hash tables BCH consistently outperforms all the baseline methods with a considerable performance gaps over them. This observation indicates that even the similarity matrix is computed based on Euclidean distance, the multiple tables of BCH can jointly approximate the complex semantic similarities better than other methods.

Table 1 also reports the training time of different multi-table methods on SIFT-1M. Among all complementary multi-table methods, MVCH consumes much more time

[2]https://www.cs.toronto.edu/~kriz/cifar.html

[3]lms.comp.nus.edu.sg/research/NUS-WIDE.htm

than others, which is partially because it heavily relies on the expensive exemplar-based feature transformation. Compared to BS and CH, our BCH takes comparable time and is able to learn multiple hash tables in an efficient way.

Conclusions

Motivated by the fact that multi-table search can be regarded as an aggregation of neighbor predictions stemming from multiple tables, we proposed a boosted complementary hash-tables (BCH) method that jointly optimizes multiple hash tables in a boosting framework. Based on the idea of ensemble learning, we devised a sequential table learning algorithm, and guaranteed the table complementarity for the maximal coverage of the nearest neighbors. To further improve the discriminative power of each table, we introduced an discrete alternating optimization algorithm to directly pursue the near-optimal binary codes. The significant performance gains of our proposed method over several large-scale benchmarks encourage the future study on joint multi-table learning with strong complementarity.

Acknowledgments

This work was supported by the National Natural Science Foundation of China (61402026 and 61572388), and the Fund of State Key Lab of Software Development Environment (SKLSDE-2016ZX-04).

References

Cheng, J.; Leng, C.; Wu, J.; Cui, H.; and Lu, H. 2014. Fast and accurate image matching with cascade hashing for 3d reconstruction. In *IEEE CVPR*, 4321–4328.

Datar, M.; Immorlica, N.; Indyk, P.; and Mirrokni, V. S. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *SCG*, 253–262.

Deng, C.; Deng, H.; Liu, X.; and Yuan, Y. 2015. Adaptive multi-bit quantization for hashing. *Neurocomputing* 151, Part 1:319 – 326.

Friedman, J.; Hastie, T.; and Tibshirani, R. 1998. Additive logistic regression: a statistical view of boosting. *Annals of Statistics* 28:2000.

Gong, Y., and Lazebnik, S. 2011. Iterative quantization: A procrustean approach to learning binary codes. In *IEEE CVPR*, 817–824.

Gong, Y.; Kumar, S.; Verma, V.; and Lazebnik, S. 2012. Angular quantization-based binary codes for fast similarity search. In *NIPS*, 1205–1213.

He, J.; Feng, J.; Liu, X.; Cheng, T.; Lin, T.-H.; Chung, H.; and Chang, S.-F. 2012. Mobile product search with bag of hash bits and boundary reranking. In *IEEE CVPR*, 3005–3012.

Kang, W.-C.; Li, W.-J.; and Zhou, Z.-H. 2016. Column sampling based discrete supervised hashing. In *AAAI*, 2604–2623.

Kong, W., and Li, W.-J. 2012. Double-bit quantization for hashing. In *AAAI*, 2604–2623.

Li, Z.; Liu, X.; Wu, J.; and Su, H. 2016. Adaptive binary quantization for fast nearest neighbor search. In *ECAI*, 64–72.

Lin, G.; Shen, C.; and van den Hengel, A. 2015. Supervised hashing using graph cuts and boosted decision trees. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 37(11):2317–2331.

Liong, V. E.; Lu, J.; Wang, G.; Moulin, P.; and Zhou, J. 2015. Deep hashing for compact binary codes learning. In *IEEE CVPR*, 2475–2483.

Liu, W.; Wang, J.; Kumar, S.; and Chang, S.-F. 2011. Hashing with graphs. In *ICML*, 1–8.

Liu, W.; Mu, C.; Kumar, S.; and Chang, S.-F. 2014a. Discrete graph hashing. In *NIPS*, 3419–3427.

Liu, X.; He, J.; Deng, C.; and Lang, B. 2014b. Collaborative hashing. In *IEEE CVPR*, 2147–2154.

Liu, X.; Huang, L.; Deng, C.; Lu, J.; and Lang, B. 2015. Multi-view complementary hash tables for nearest neighbor search. In *IEEE ICCV*, 1107–1115.

Liu, H.; Ji, R.; Wu, Y.; and Liu, W. 2016a. Towards optimal binary code learning via ordinal embedding. In *AAAI*, 1258–1265.

Liu, X.; Fan, X.; Deng, C.; Li, Z.; Su, H.; and Tao, D. 2016b. Multilinear hyperplane hashing. In *IEEE CVPR*, 1–9.

Liu, X.; He, J.; and Lang, B. 2013. Reciprocal Hash Tables for Nearest Neighbor Search. In *AAAI*, 626–632.

Lv, Q.; Josephson, W.; Wang, Z.; Charikar, M.; and Li, K. 2007. Multi-probe lsh: efficient indexing for high-dimensional similarity search. In *VLDB*, 950–961.

Mu, Y.; Chen, X.; Liu, X.; Chua, T.-S.; and Yan, S. 2012. Multimedia semantics-aware query-adaptive hashing with bits reconfigurability. *IJMIR* 1(1):59–70.

Mu, Y.; Hua, G.; Fan, W.; and Chang, S.-F. 2014. Hash-svm: Scalable kernel machines for large-scale visual classification. In *IEEE CVPR*, 979–986.

Mu, Y.; Liu, W.; Deng, C.; Lv, Z.; and Gao, X. 2016. Fast structural binary coding. In *IJCAI*, 1860–1866.

Norouzi, M.; Punjani, A.; and Fleet, D. J. 2012. Fast search in hamming space with multi-index hashing. In *IEEE CVPR*, 3108–3115.

Shen, F.; Shen, C.; Liu, W.; and Tao Shen, H. 2015. Supervised discrete hashing. In *IEEE CVPR*, 37–45.

Slaney, M.; Lifshits, Y.; and He, J. 2012. Optimal parameters for locality-sensitive hashing. *Proceedings of the IEEE* 100(9):634–640.

Song, D.; Liu, W.; and Meyer, D. A. 2016. Coordinate discrete optimization for efficient cross-view image retrieval. In *IJCAI*, 2018–2024.

Wang, Q.; Si, L.; and Shen, B. 2015. Learning to hash on structured data. In *AAAI*, 3066–3072.

Weiss, Y.; Torralba, A.; and Fergus, R. 2008. Spectral hashing. In *NIPS*, 1–8.

Xia, Y.; He, K.; Wen, F.; and Sun, J. 2013. Joint inverted indexing. In *IEEE ICCV*, 3416–3423.

Xia, R.; Pan, Y.; Lai, H.; Liu, C.; and Yan, S. 2014. Supervised hashing for image retrieval via image representation learning. In *AAAI*, 2156–2162.

Xu, H.; Wang, J.; Li, Z.; Zeng, G.; Li, S.; and Yu, N. 2011. Complementary hashing for approximate nearest neighbor search. In *IEEE ICCV*, 1631–1638.

Yu, X.; Kumar, S.; Gong, Y.; and Chang, S. 2014. Circulant binary embedding. In *ICML*, 1–8.

Zhu, H.; Long, M.; Wang, J.; and Cao, Y. 2016. Deep hashing network for efficient similarity retrieval. In *AAAI*, 2415–2421.