Supervised Hashing with Latent Factor Models

Peichao Zhang
Shanghai Key Laboratory of Scalable Computing and Systems
Department of Computer Science and Engineering
Shanghai Jiao Tong University, China
starforever00@gmail.com

Wei Zhang
Shanghai Key Laboratory of Scalable Computing and Systems
Department of Computer Science and Engineering
Shanghai Jiao Tong University, China
razhangwei@gmail.com

Wu-Jun Li
National Key Laboratory for Novel Software Technology
Department of Computer Science and Technology
Nanjing University, China
liwujun@nju.edu.cn

Minyi Guo
Shanghai Key Laboratory of Scalable Computing and Systems
Department of Computer Science and Engineering
Shanghai Jiao Tong University, China
guo-my@cs.sjtu.edu.cn

ABSTRACT

Due to its low storage cost and fast query speed, hashing has been widely adopted for approximate nearest neighbor search in large-scale datasets. Traditional hashing methods try to learn the hash codes in an *unsupervised* way where the *metric (Euclidean) structure* of the training data is preserved. Very recently, *supervised* hashing methods, which try to preserve the *semantic structure* constructed from the *semantic labels* of the training points, have exhibited higher accuracy than unsupervised methods. In this paper, we propose a novel supervised hashing method, called *latent factor hashing* (LFH), to learn similarity-preserving binary codes based on latent factor models. An algorithm with convergence guarantee is proposed to learn the parameters of LFH. Furthermore, a linear-time variant with stochastic learning optimization is proposed for training LFH on large-scale datasets. Experimental results on two large datasets with semantic labels show that LFH can achieve superior accuracy than state-of-the-art methods with comparable training time.

Categories and Subject Descriptors

H.3.3 [**Information Storage And Retrieval**]: Information Search and Retrieval—*Retrieval models*

SIGIR'14, July 6–11, 2014, Gold Coast, Queensland, Australia.

http://dx.doi.org/10.1145/2600428.2609600 .

Keywords

Hashing; Latent Factor Model; Image Retrieval; Big Data

1. INTRODUCTION

Nearest neighbor (NN) search plays a fundamental role in machine learning and related areas, such as pattern recognition, information retrieval, data mining and computer vision. In many real applications, it's not necessary for an algorithm to return the exact nearest neighbors for every possible query. Hence, in recent years approximate nearest neighbor (ANN) search algorithms with improved speed and memory saving have been received more and more attention by researchers [1, 2, 7].

Over the last decades, there has been an explosive growth of data from many areas. To meet the demand of performing ANN search on these massive datasets, various hashing techniques have been proposed due to their fast query speed and low storage cost [1, 4, 5, 8, 10, 16, 20, 21, 26, 27, 31, 35, 36, 37, 38, 39, 40, 41]. The essential idea of hashing is to map the data points from the original feature space into binary codes in the hashcode space with similarities between pairs of data points preserved. Hamming distance is used to measure the closeness between binary codes, which is defined as the number of positions at which two binary codes differ. More specifically, when two data points are deemed as similar, their binary codes should have a low Hamming distance. On the contrary, when two data points are dissimilar, a high Hamming distance is expected between their binary codes. The advantage of binary codes representation over the original feature vector representation is twofold. Firstly, each dimension of a binary code can be stored using only 1 bit while several bytes are typically required for one dimension of the original feature vector, leading to a dramatic reduction in storage cost. Secondly, by using binary codes representation, all the data points within a specific Hamming distance to a given query can be retrieved

in constant or sub-linear time regardless of the total size of the dataset [30]. Because of these two advantages, hashing techniques have become a promising choice for efficient ANN search on massive datasets.

Existing hashing methods can be divided into two categories: data-independent methods and data-dependent methods [6, 17, 18]. For data-independent methods, the hashing functions are learned without using any training data. Representative data-independent methods include locality-sensitive hashing (LSH) [1, 5, 7], shift-invariant kernels hashing (SIKH) [22], and many other extensions [4, 13, 14]. On the other hand, for data-dependent methods, their hashing functions are learned from some training data. Generally speaking, data-dependent methods often require less number of bits than data-independent methods to achieve satisfactory performance.

The data-dependent methods can be further divided into two categories: unsupervised and supervised methods [17, 20, 32]. Unsupervised methods try to preserve the metric (Euclidean) structure between data points using only their feature information. Representative unsupervised methods include spectral hashing (SH) [34], principal component analysis based hashing (PCAH) [33], iterative quantization (ITQ) [6], anchor graph hashing (AGH) [18], isotropic hashing (IsoHash) [9], multimodel latent binary embedding (MLBE) [42] and predictable dual-view hashing (PDH) [23]. Due to the fact that high level semantic description of an object often differs from the extracted low level feature descriptors, known as semantic gap [25], returning nearest neighbors according to metric distances between the feature vectors doesn't always guarantee a good search quality. Hence, many recent works focus on supervised methods which try to preserve the semantic structure among the data points by utilizing their associated semantic information [17, 19]. Although there are also some works to exploit other types of supervised information like the ranking information for hashing [16, 20], the semantic information is usually given in the form of pairwise labels indicating whether two data points are known to be similar or dissimilar. Representative supervised methods include restricted Boltzmann machine based hashing (RBM) [24], binary reconstructive embedding (BRE) [12], sequential projection learning for hashing (SPLH) [33], minimal loss hashing (MLH) [19], kernel-based supervised hashing (KSH) [17], and linear discriminant analysis based hashing (LDAHash) [28]. Additionally, there are also some semi-supervised hashing methods [32] which use both labeled data and unlabeled data to train their model. As stated in recent works [17, 19, 20], sophisticated supervised methods, such as SPLH, MLH, and KSH, can achieve higher accuracy than unsupervised methods. However, some existing supervised methods, like MLH, suffer from a very large amount of training time, making it difficult to apply to large-scale datasets.

In this paper, we propose a novel method, called latent factor hashing (LFH), for supervised hashing. The main contributions of this paper are outlined as follows:

- Base on latent factor models, the likelihood of the pairwise similarities are elegantly modeled as a function of the Hamming distance between the corresponding data points.
- An algorithm with convergence guarantee is proposed to learn the parameters of LFH.
- To model the large-scale problems, a linear-time variant with stochastic learning optimization is proposed for fast parameter learning.
- Experimental results on two real datasets with semantic labels show that LFH can achieve much higher accuracy than other state-of-the-art methods with efficiency in training time.

The rest of the this paper is organized as follows: In Section 2, we will introduce the details of our LFH model. Experimental results are presented in Section 3. Finally, we conclude the paper in Section 4.

2. LATENT FACTOR HASHING

In this section, we present the details of our latent factor hashing (LFH) model, including the model formulation and learning algorithms.

2.1 Problem Definition

Suppose we have N points as the training data, each represented as a feature vector $\mathbf{x}_i \in \mathbb{R}^D$. Some pairs of points have similarity labels s_{ij} associated, where $s_{ij} = 1$ means $\mathbf{x}_i$ and $\mathbf{x}_j$ are similar and $s_{ij} = 0$ means $\mathbf{x}_i$ and $\mathbf{x}_j$ are dissimilar. Our goal is to learn a binary code $\mathbf{b}_i \in \{-1, 1\}^Q$ for each $\mathbf{x}_i$ with similarity between pairs preserved. In particular, when $s_{ij} = 1$, the binary codes $\mathbf{b}_i$ and $\mathbf{b}_j$ should have a low Hamming distance. On the other hand, when $s_{ij} = 0$, the Hamming distance between $\mathbf{b}_i$ and $\mathbf{b}_j$ should be high. In compact form, we use a matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$ to denote all the feature vectors, a matrix $\mathbf{B} \in \{-1, 1\}^{N \times Q}$ to denote all the binary codes, and a set $\mathcal{S} = \{s_{ij}\}$ to denote all the observed similarity labels. Additionally, we use the notation $\mathbf{A}_{i*}$ and $\mathbf{A}_{*j}$ to denote the ith row and the jth column of a matrix $\mathbf{A}$, respectively. $\mathbf{A}^T$ is the transpose of $\mathbf{A}$. The similarity labels $\mathcal{S}$ can be constructed from the neighborhood structure by thresholding on the metric distances between the feature vectors [17]. However, such $\mathcal{S}$ is of low quality since no semantic information is utilized. In supervised hashing setting, $\mathcal{S}$ is often constructed from the semantic labels within the data points. Such labels are often built with manual effort to ensure its quality.

2.2 Model Formulation

Let Θ_{ij} denote half of the inner product between two binary codes $\mathbf{b}_i, \mathbf{b}_j \in \{-1, 1\}^Q$:

$$\Theta_{ij} = \frac{1}{2}\mathbf{b}_i^T \mathbf{b}_j.$$

The likelihood of the observed similarity labels $\mathcal{S}$ can be defined as follows:

$$p(\mathcal{S} \mid \mathbf{B}) = \prod_{s_{ij} \in \mathcal{S}} p(s_{ij} \mid \mathbf{B}), \tag{1}$$

with

$$p(s_{ij} \mid \mathbf{B}) = \begin{cases} a_{ij}, & s_{ij} = 1 \\ 1 - a_{ij}, & s_{ij} = 0 \end{cases},$$

where $a_{ij} = \sigma(\Theta_{ij})$ with σ being the logistic function:

$$\sigma(x) = \frac{1}{1 + e^{-x}}.$$

It is easy to prove the following relationship between the Hamming distance $\text{dist}_H(\cdot,\cdot)$ and inner product of two binary codes:

$$\text{dist}_H(\mathbf{b}_i, \mathbf{b}_j) = \frac{1}{2}(Q - \mathbf{b}_i^T \mathbf{b}_j) = \frac{1}{2}(Q - 2\Theta_{ij}).$$

We can find that the smaller the $\text{dist}_H(\mathbf{b}_i, \mathbf{b}_j)$ is, the larger $p(s_{ij} = 1 \mid \mathbf{B})$ will be. Maximizing the likelihood of $\mathcal{S}$ in (1) will make the Hamming distance between two similar points as small as possible, and that between two dissimilar points as high as possible. Hence this model is reasonable and matches the goal to preserve similarity.

With some prior $p(\mathbf{B})$, the posteriori of $\mathbf{B}$ can be computed as follows:

$$p(\mathbf{B} \mid \mathcal{S}) \sim p(\mathcal{S} \mid \mathbf{B}) p(\mathbf{B}).$$

We can use maximum a posteriori estimation to learn the optimal $\mathbf{B}$. However, directly optimizing on $\mathbf{B}$ is an NP-hard problem [34]. Following most existing hashing methods, we compute the optimal $\mathbf{B}$ through two stages. In the first stage, we relax $\mathbf{B}$ to be a real valued matrix $\mathbf{U} \in \mathbb{R}^{N \times Q}$. The ith row of $\mathbf{U}$ is called the latent factor for the ith data point. We learn an optimal $\mathbf{U}$ under the same probabilistic framework as for $\mathbf{B}$. Then in the second stage, we perform some rounding technique on the real valued $\mathbf{U}$ to get the binary codes $\mathbf{B}$.

More specifically, we replace all the occurrences of $\mathbf{B}$ in previous equations with $\mathbf{U}$. Θ_{ij} is then re-defined as:

$$\Theta_{ij} = \frac{1}{2}\mathbf{U}_{i*}\mathbf{U}_{j*}^T.$$

Similarly, $p(\mathcal{S} \mid \mathbf{B})$, $p(\mathbf{B})$, $p(\mathbf{B} \mid \mathcal{S})$ are replaced with $p(\mathcal{S} \mid \mathbf{U})$, $p(\mathbf{U})$, $p(\mathbf{U} \mid \mathcal{S})$, respectively. We put a normal distribution on $p(\mathbf{U})$:

$$p(\mathbf{U}) = \prod_{d=1}^{Q} \mathcal{N}(\mathbf{U}_{*d} \mid \mathbf{0}, \beta\mathbf{I}),$$

where $\mathcal{N}(\cdot)$ denotes the normal distribution, $\mathbf{I}$ is an identity matrix, and β is a hyper-parameter. The log posteriori of $\mathbf{U}$ can then be derived as:

$$\begin{aligned} L &= \log p(\mathbf{U} \mid \mathcal{S}) \\ &= \sum_{s_{ij} \in \mathcal{S}} (s_{ij}\Theta_{ij} - \log(1 + e^{\Theta_{ij}})) - \frac{1}{2\beta}\|\mathbf{U}\|_F^2 + c, \end{aligned} \tag{2}$$

where $\|\cdot\|_F$ denotes the Frobenius norm of a matrix, and c is a constant term independent of $\mathbf{U}$. The next step is to learn the optimal $\mathbf{U}$ that maximizes L in (2).

2.3 Learning

Since directly optimizing the whole $\mathbf{U}$ can be very time-consuming, we optimize each row of $\mathbf{U}$ at a time with its other rows fixed. We adopt the surrogate algorithm [15] to optimize each row $\mathbf{U}_{i*}$. The surrogate learning algorithm can be viewed as a generalization of the expectation-maximization (EM) algorithm. It constructs a lower bound of the objective function, and then updates the parameters to maximize that lower bound. Just like EM algorithm, we need to derive different lower bounds and optimization procedures for different models [15]. In the following content, we will derive the details of the surrogate learning algorithm for our model.

The gradient vector and the Hessian matrix of the objective function L defined in (2) with respect to $\mathbf{U}_{i*}$ can be derived as:

$$\begin{aligned} \frac{\partial L}{\partial \mathbf{U}_{i*}^T} &= \frac{1}{2}\sum_{j:s_{ij}\in\mathcal{S}} (s_{ij} - a_{ij})\mathbf{U}_{j*}^T \\ &\quad + \frac{1}{2}\sum_{j:s_{ji}\in\mathcal{S}} (s_{ji} - a_{ji})\mathbf{U}_{j*}^T - \frac{1}{\beta}\mathbf{U}_{i*}^T, \end{aligned}$$

$$\begin{aligned} \frac{\partial^2 L}{\partial \mathbf{U}_{i*}^T \partial \mathbf{U}_{i*}} &= -\frac{1}{4}\sum_{j:s_{ij}\in\mathcal{S}} a_{ij}(1 - a_{ij})\mathbf{U}_{j*}^T\mathbf{U}_{j*} \\ &\quad - \frac{1}{4}\sum_{j:s_{ji}\in\mathcal{S}} a_{ji}(1 - a_{ji})\mathbf{U}_{j*}^T\mathbf{U}_{j*} - \frac{1}{\beta}\mathbf{I}. \end{aligned}$$

If we define $\mathbf{H}_i$ as:

$$\mathbf{H}_i = -\frac{1}{16}\sum_{j:s_{ij}\in\mathcal{S}} \mathbf{U}_{j*}^T\mathbf{U}_{j*} - \frac{1}{16}\sum_{j:s_{ji}\in\mathcal{S}} \mathbf{U}_{j*}^T\mathbf{U}_{j*} - \frac{1}{\beta}\mathbf{I}, \tag{3}$$

we can prove that

$$\frac{\partial^2 L}{\partial \mathbf{U}_{i*}^T \partial \mathbf{U}_{i*}} \succeq \mathbf{H}_i,$$

where $\mathbf{A} \succeq \mathbf{B}$ means $\mathbf{A} - \mathbf{B}$ is a positive semi-definite matrix.

Then we can construct the lower bound of $L(\mathbf{U}_{i*})$, denoted by $\widetilde{L}(\mathbf{U}_{i*})$, as:

$$\begin{aligned} \widetilde{L}(\mathbf{U}_{i*}) &= L(\mathbf{U}_{i*}(t)) + (\mathbf{U}_{i*} - \mathbf{U}_{i*}(t))\frac{\partial L}{\partial \mathbf{U}_{i*}^T}(t) \\ &\quad + \frac{1}{2}(\mathbf{U}_{i*} - \mathbf{U}_{i*}(t))\mathbf{H}_i(t)(\mathbf{U}_{i*} - \mathbf{U}_{i*}(t))^T. \end{aligned}$$

The values of $\mathbf{U}$ and other parameters that depend on $\mathbf{U}$ change through the updating process. Here we use the notation $x(t)$ to denote the value of a parameter x at some iteration t. We update $\mathbf{U}_{i*}$ to be the one that gives the maximum value of $\widetilde{L}(\mathbf{U}_{i*})$. It is easy to see that $\widetilde{L}(\mathbf{U}_{i*})$ is a quadratic form in the variable $\mathbf{U}_{i*}$, which can be proved to be convex. Hence, we can find the optimum value of $\mathbf{U}_{i*}$ by setting the gradient of $\widetilde{L}(\mathbf{U}_{i*})$ with respect to $\mathbf{U}_{i*}$ to 0. As a result, we end up with the following update rule for $\mathbf{U}_{i*}$:

$$\mathbf{U}_{i*}(t+1) = \mathbf{U}_{i*}(t) - [\frac{\partial L}{\partial \mathbf{U}_{i*}^T}(t)]^T \mathbf{H}_i(t)^{-1}. \tag{4}$$

We can then update other rows of $\mathbf{U}$ iteratively using the above rule.

The convergence of the iterative updating process is controlled by some threshold value ε and the maximum allowed number of iterations T. Here ε and T are both hyper-parameters. During each iteration, we update $\mathbf{U}$ by updating each of its rows sequentially. The initial value of $\mathbf{U}$ can be obtained through PCA on the feature space $\mathbf{X}$. The pseudocode of the updating process is shown in Algorithm 1.

2.3.1 Rounding

After the optimal $\mathbf{U}$ is learned, we can obtain the final binary codes $\mathbf{B}$ using some rounding techniques. In this paper, to keep our method simple, the real values of $\mathbf{U}$ are quantized into the binary codes of $\mathbf{B}$ by taking their signs, that is:

$$B_{ij} = \begin{cases} 1, & U_{ij} > 0 \\ -1, & \text{otherwise} \end{cases}.$$

Algorithm 1 Optimizing $\mathbf{U}$ using surrogate algorithm

Input: $\mathbf{X} \in \mathbb{R}^{N\times D}, \mathcal{S} = \{s_{ij}\}, Q, T \in \mathbb{N}^+, \beta, \varepsilon \in \mathbb{R}^+$.
Initializing $\mathbf{U}$ by performing PCA on $\mathbf{X}$.
for $t = 1 \to T$ **do**
  for $i = 1 \to N$ **do**
    Update $\mathbf{U}_{i*}$ by following the rule given in (4).
  end for
  Compute L in (2) using the updated $\mathbf{U}$.
  Terminate the iterative process when the change of L is smaller than ε.
end for
Output: $\mathbf{U} \in \mathbb{R}^{N\times Q}$.

2.3.2 *Out-of-Sample Extension*

In order to perform ANN search, we need to compute the binary code $\mathbf{b}$ for a query $\mathbf{x}$ which is not in the training set. We achieve this by finding a matrix $\mathbf{W} \in \mathbb{R}^{D\times Q}$ that maps $\mathbf{x}$ to $\mathbf{u}$ in the following way:

$$\mathbf{u} = \mathbf{W}^T\mathbf{x}.$$

We then perform the same rounding technique discussed in Section 2.3.1 on $\mathbf{u}$ to obtain $\mathbf{b}$.

We use linear regression to train $\mathbf{W}$ over the training set. The squared loss with regularization term is shown below:

$$L_e = \|\mathbf{U} - \mathbf{X}\mathbf{W}\|_F^2 + \lambda_e\|\mathbf{W}\|_F^2.$$

And the optimal $\mathbf{W}$ can be computed as:

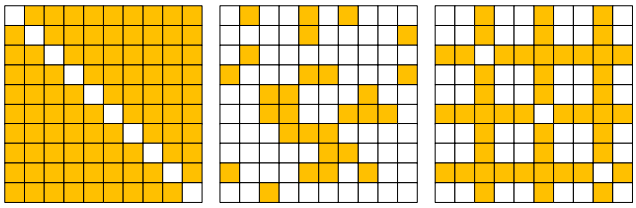
$$\mathbf{W} = (\mathbf{X}^T\mathbf{X} + \lambda_e\mathbf{I})^{-1}\mathbf{X}^T\mathbf{U}. \tag{5}$$

2.4 Convergence and Complexity Analysis

At each iteration, we first construct a lower bound at the current point $\mathbf{U}_{i*}(t)$, and then optimize it to get a new point $\mathbf{U}_{i*}(t+1)$ with a higher function value $L(\mathbf{U}_{i*}(t+1))$. Hence, the objective function value always increases in the new iteration. Furthermore, the objective function value $L = \log p(\mathbf{U} \mid \mathcal{S})$ is upper bounded by zero. Hence, our Algorithm 1 can always guarantee convergence to local maximum, the principle of which is similar to that of EM algorithm. This convergence property will also be illustrated in Figure 2. The convergence property of our surrogate algorithm is one of the key advantages compared with standard Newton method and first-order gradient method. In both Newton method and first-order gradient method, a step size, also called learning rate in some literatures, should be manually specified, which cannot necessarily guarantee convergence.

We can prove that, when updating $\mathbf{U}_{i*}$, the total time to compute $\partial L/\partial \mathbf{U}_{i*}^T$ and $\mathbf{H}_i$ for all $i = 1, \ldots, N$ is $\mathcal{O}(|\mathcal{S}|Q)$ and $\mathcal{O}(|\mathcal{S}|Q^2)$, respectively. Since the inversion of $\mathbf{H}_i$ can be computed in $\mathcal{O}(Q^3)$, the time to update $\mathbf{U}_{i*}$ following the rule given in (4) is $\mathcal{O}(Q^3)$. Then the time to update $\mathbf{U}$ by one iteration is $\mathcal{O}(NQ^3 + |\mathcal{S}|Q^2)$. Therefore, the total time of the updating process is $\mathcal{O}(T(NQ^3 + |\mathcal{S}|Q^2))$, where T is the number of iterations.

Besides that, the time for rounding is $\mathcal{O}(NQ)$. And the time to compute $\mathbf{W}$ for out-of-sample extension is $\mathcal{O}(ND^2 + D^3 + NDQ)$, which can be further reduced to $\mathcal{O}(ND^2)$ with the typical assumption that $Q < D \ll N$. With the precomputed $\mathbf{W}$, the out-of-sample extension for a query $\mathbf{x}$ can be achieved in $\mathcal{O}(DQ)$.



(a) Full (b) Sparse (c) Aligned

Figure 1: Selection of $\mathcal{S}$ for stochastic learning.

2.5 Stochastic Learning

For a given set of N training points with supervised information, there are $N \times N$ pairs of similarity labels available to form $\mathcal{S}$. Straightforwardly, we can choose $\mathcal{S}$ to include all the available similarity pairs, that is, $\mathcal{S} = \{s_{ij} \mid i, j = 1, \ldots, N, i \neq j\}$. This is illustrated in Figure 1(a), in which a colored cell in the i-th row and j-th column indicates that s_{ij} is included in $\mathcal{S}$. In this way, the best possible accuracy can be achieved since we use as much as available semantic information. However, according to the complexity analysis described in Section 2.4, if we set $\mathcal{S}$ to contain the full supervised information, the time cost to update $\mathbf{U}$ would become $\mathcal{O}(NQ^3 + N^2Q^2)$ per iteration, and $\mathcal{O}(N^2)$ memory is needed to store $\mathcal{S}$. Such cost in both computation and storage is unacceptable when N grows large.

Inspired by stochastic gradient descent method, we propose an efficient way of updating $\mathbf{U}$, called stochastic learning. As illustrated in Figure 1(b), $\mathcal{S}$ contains a sparse subset with only $\mathcal{O}(NQ)$ entries which are randomly selected from all the available similarity pairs. The random sampling is performed for each iteration. We choose the size of $\mathcal{S}$ to be $\mathcal{O}(NQ)$ so that the time cost to update $\mathbf{U}$ is reduced to $\mathcal{O}(NQ^3)$ per iteration. For storage efficiency, we compute only the sampled subset during each iteration. By this way, the maximum required storage reduces to $\mathcal{O}(NQ)$.

We can even further reduce the time cost by sampling $\mathcal{S}$ in an aligned way. During each iteration, an index set $\mathcal{I}$ of size $\mathcal{O}(Q)$ is randomly chosen from $\{1, \ldots, N\}$. We then construct $\mathcal{S}$ by using the rows and columns in $\mathcal{I}$, with entries at the diagonal excluded. The resulted $\mathcal{S}$ is shown in Figure 1(c). Following this, the constructed $\mathcal{S}$ is guaranteed to be symmetric, and $\mathbf{H}_i$ defined in (3) can be simplified as:

$$\mathbf{H}_i = -\frac{1}{8}\sum_{j:s_{ij}\in\mathcal{S}} \mathbf{U}_{j*}^T\mathbf{U}_{j*} - \frac{1}{\beta}\mathbf{I}.$$

For all $i \notin \mathcal{I}$, the set $\{j : s_{ij} \in \mathcal{S}\}$ is exactly the same thanks to the alignment of $\mathcal{S}$. This implies that $\mathbf{H}_i$ remains the same for all $i \notin \mathcal{I}$. Therefore, we can compute $\mathbf{H}_i^{-1}$ in a preprocessing step. By doing so, the time cost to update $\mathbf{U}_{i*}$ for each $i \notin \mathcal{I}$ can be reduced to $\mathcal{O}(Q^2)$. For each $i \in \mathcal{I}$, we can update $\mathbf{U}_{i*}$ through some complicated calculations while still maintaining the $\mathcal{O}(Q^2)$ time complexity. We can also safely skip updating $\mathbf{U}_{i*}$ for that iteration without much loss in performance due to the fact that Q is much smaller than N. Even though $\mathbf{U}_{i*}$ is not updated for some small portion of i in one single iteration, it will much likely be updated in the subsequent iterations because $\mathcal{I}$ changes among the iterations. As a result, the total time cost to update $\mathbf{U}$ is reduced to $\mathcal{O}(NQ^2)$ per iteration. For Q up to 128, this makes our learning process two orders of magnitude faster.

Consequently, since Q is bounded by a small constant, we can say that the cost in computation and storage of our learning algorithm are linear to the number of training points N. This makes our LFH easily scalable to very large datasets.

2.6 Normalized Hyper-parameters

The hyper-parameter β in the objective function defined in (2) acts as a factor weighing the relative importance between the first and the second term. However, the number of sum items in each term is different: in the first term there are $|\mathcal{S}|$ sum items, while in the second term there are N sum items. Since different datasets may have different values of N and $|\mathcal{S}|$, the optimal value of β may vary between the datasets. To address this issue and make our method less dependent on a specific dataset, we introduce a new hyper-parameter β' satisfying that:

$$\beta' = \frac{N}{|\mathcal{S}|}\beta.$$

By replacing β with β' in (2), we have a specialized parameter β' for each dataset. The optimal value for β is then normalized to roughly the same range on different datasets. We can normalize λ_ϵ in (5) by following the same idea.

We find that the MLH method spends most of the time on selecting the best hyper-parameters for each dataset. With the normalized hyper-parameters introduced, we can pre-compute the optimal values for the hyper-parameters on some smaller dataset, and then apply the same values to all other datasets. This saves us the time of hyper-parameter selection and makes our method more efficient on large datasets.

3. EXPERIMENT

3.1 Datasets

We evaluate our method on two standard large image datasets with semantic labels: CIFAR-10 [11] and NUS-WIDE [3].

The CIFAR-10 dataset [11] consists of 60,000 color images drawn from the 80M tiny image collection [29]. Each image of size 32×32 is represented by a 512-dimensional GIST feature vector. Each image is manually classified into one of the 10 classes, with an exclusive label indicating its belonging class. Two images are considered as a ground truth neighbor if they have the same class label.

The NUS-WIDE dataset [3] contains 269,648 images collected from Flickr. Each image is represented by a 1134-dimensional low level feature vector, including color histogram, color auto-correlogram, edge direction histogram, wavelet texture, block-wise color moments, and bag of visual words. The images are manually assigned with some of the 81 concept tags. The ground truth neighbor is defined on two images if they share at least one common tag.

For data pre-processing, we follow the standard way of feature normalization by making each dimension of the feature vectors to have zero mean and equal variance.

3.2 Experimental Settings and Baselines

For both CIFAR-10 and NUS-WIDE datasets, we randomly sample 1,000 points as query set, 1,000 points as validation set, and all the remaining points as training set. Using normalized hyper-parameters described in Section 2.6, the best hyper-parameters are selected by using the validation set of CIFAR-10. All experimental results are averaged over 10 independent rounds of random training / validation / query partitions.

Unless otherwise stated, we refer LFH in the experiment section to the LFH with stochastic learning. We compare our LFH method with some state-of-the-art hashing methods, which include:

- Data-independent methods: locality-sensitive hashing (LSH), and shift-invariant kernels hashing (SIKH);
- Unsupervised data-dependent methods: spectral hashing (SH), principal component analysis based hashing (PCAH), iterative quantization (ITQ), anchor graph hashing (AGH);
- Supervised data-dependent methods: sequential projection learning for hashing (SPLH), minimal loss hashing (MLH), and kernel-based supervised hashing (KSH).

All the baseline methods are implemented using the source codes provided by the corresponding authors. For KSH and AGH, the number of support points for kernel construction is set to 300 by following the same settings in [17, 18]. For KSH, SPLH, and MLH, it's impossible to use all the supervised information for training since it would be very time-consuming. Following the same strategy used in [17], we sample 2,000 labeled points for these methods.

All our experiments are conducted on a workstation with 24 Intel Xeon CPU cores and 64 GB RAM.

3.3 Effect of Stochastic Learning

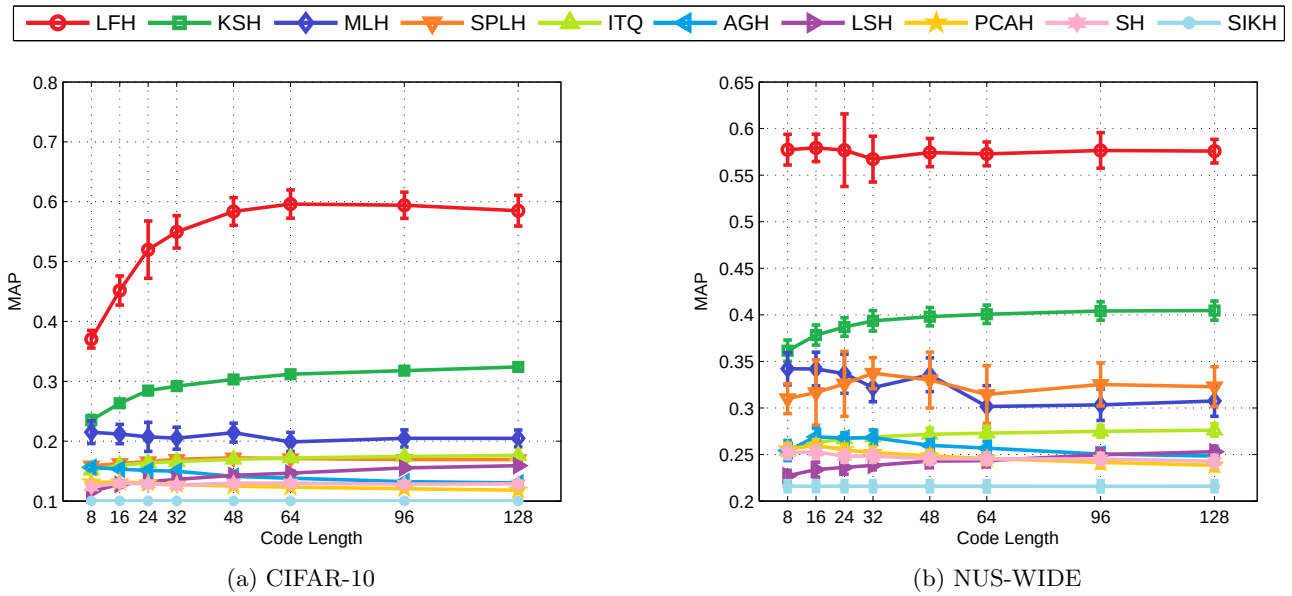
The convergence curve of the objective function on a sampled CIFAR-10 subset of 5000 points with code length 32 is shown in Figure 2. The LFH-Full method refers to the LFH that uses the full supervised information for updating, and LFH-Stochastic refers to the LFH with stochastic learning. The objective function value is computed based on the full supervised information for both methods. We can see that the objective function of LFH-Full converges to a stationary point after a few iterations. The objective function of LFH-Stochastic has a major trend of convergence to some stationary point with slight vibration. This behavior is quite similar to stochastic gradient descent method and is empirically acceptable.

Figure 3 shows the mean average precision (MAP) [10, 17, 19] values computed on a validation set during the updating process. The final MAP evaluated on a query set is 0.5237 for LFH-Full and 0.4694 for LFH-Stochastic. The reduction in MAP of LFH-Stochastic is affordable given the dramatic decrease in time complexity by using stochastic learning.

3.4 Hamming Ranking Performance

We perform Hamming ranking using the generated binary codes on the CIFAR-10 and NUS-WIDE datasets. For each query in the query set, all the points in the training set are ranked according to the Hamming distance between their binary codes and the query's. The MAP is reported to evaluate the accuracy of different hashing methods.

Figure 4(a) and Figure 4(b) show the averaged MAP results with different code lengths on the two datasets, respectively. We can find that with the help of semantic information, supervised data-dependent methods can generally



(a) CIFAR-10 (b) NUS-WIDE

Figure 4: MAP results with different code lengths.

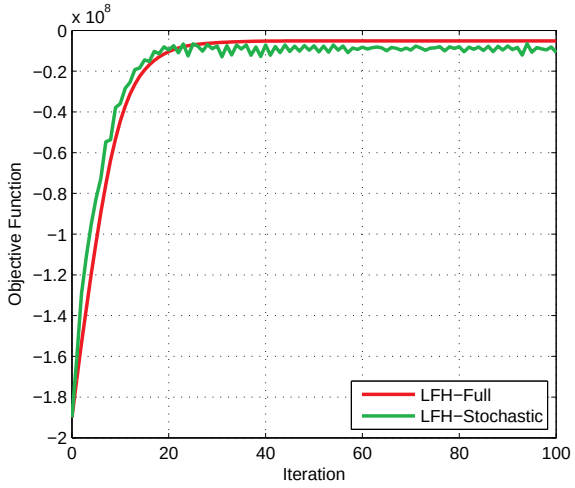


Figure 2: Convergence curve.

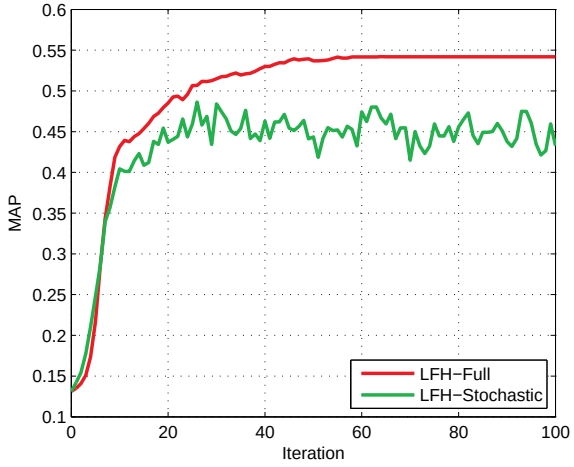


Figure 3: MAP during the iterations.

achieve better accuracy than data-independent and unsupervised data-dependent methods. Furthermore, the accuracy of our LFH method is much higher than other methods including these supervised data-dependent methods KSH, SPLH, and MLH.

The precision-recall curves with different code lengths will be illustrated in the Appendix at the end of this paper (refer to Figure 9 and Figure 10), which will also show that our LFH method can significantly outperform other state-of-the-art hashing methods.

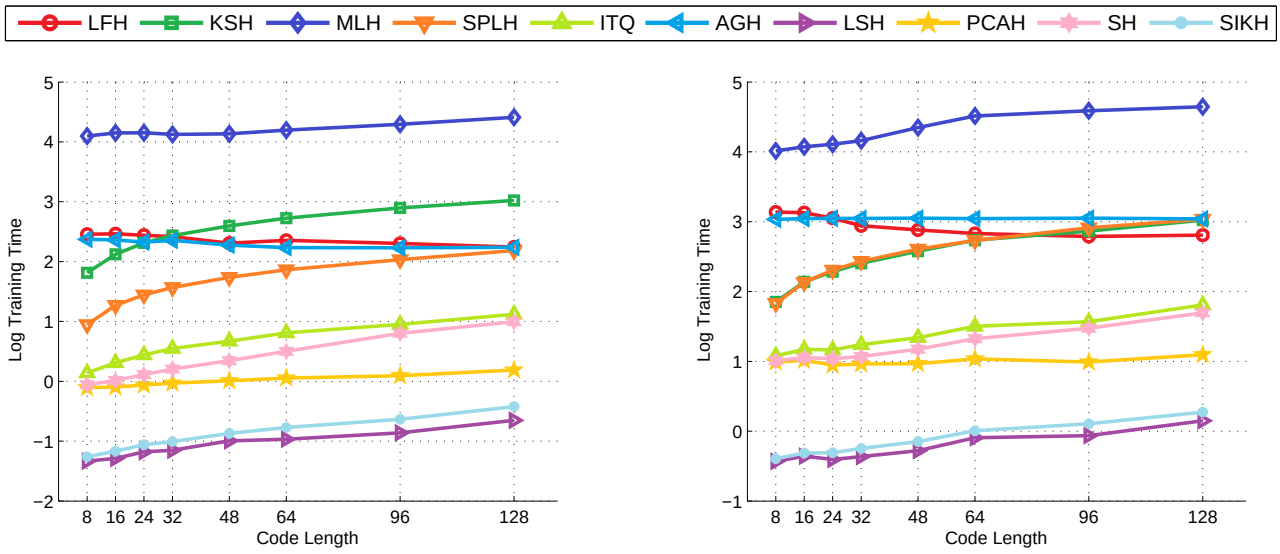
3.5 Computational Cost

Figure 5(a) and Figure 5(b) show the average training time of different hashing methods with different code lengths on the CIFAR-10 and NUS-WIDE datasets, respectively. The reported values are in seconds in a logarithmic scale.

We can find that the data-independent hashing methods require the least amount of training time, and the supervised data-dependent hashing methods need the most amount of training time. Compared to other supervised data-dependent hashing methods, the training time of LFH is much smaller than that of MLH and is comparable to that of KSH and SPLH. For large code lengths, our LFH is even faster than KSH and SPLH. This is because the number of iterations needed to learn $\mathbf{U}$ decreases as the code length increases.

3.6 Performance using Full Supervised Information

For the results reported in Section 3.4 and Section 3.5, we adopt the same strategy as that in [17] to train KSH, SPLH, and MLH by sampling only 2,000 labeled points due to their high time complexity. To get a deeper comparison,



(a) CIFAR-10 (b) NUS-WIDE

Figure 5: Training time with different code lengths.

we perform another experiment on smaller datasets where the full supervised information can be used for training. We randomly sample a subset of CIFAR-10 with 5000 points for evaluation. We also include LFH with stochastic learning to better demonstrate its effectiveness. Figure 6 and Figure 7 show the accuracy and computational cost for these methods.

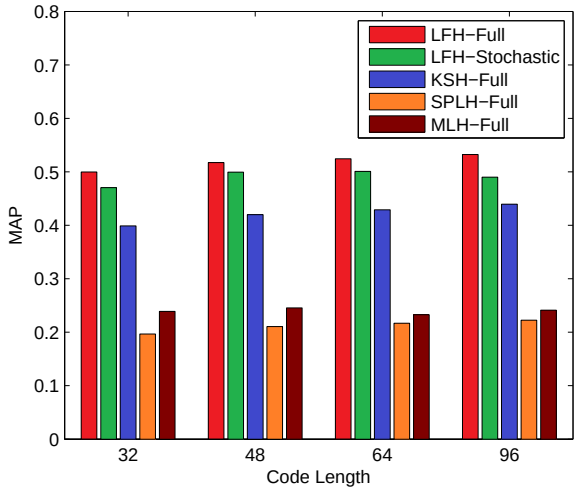


Figure 6: Accuracy on CIFAR-10 subset with full labels.

We can see that our LFH, even with stochastic learning, can achieve higher MAP than other methods with full labels used. The training speed of LFH with full labels is comparable to that of KSH and SPLH, and is much faster than that of MLH. The LFH with stochastic learning beats all other methods in training time.

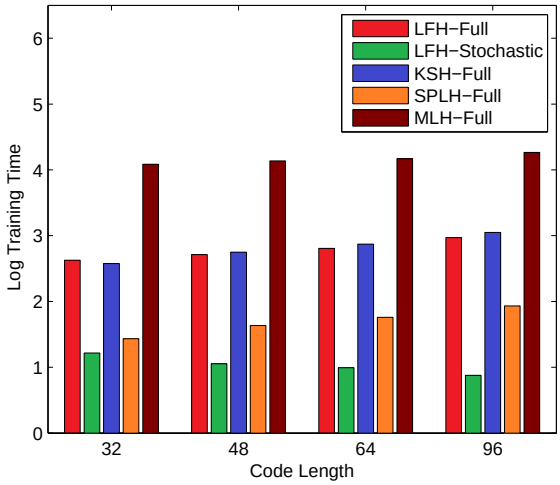


Figure 7: Computational cost on CIFAR-10 subset with full labels.

Hence, we can conclude that our LFH method can outperform other supervised hashing methods in terms of both accuracy and computational cost.

3.7 Case Study

In Figure 8, we demonstrate the hamming ranking results for some example queries on the CIFAR-10 dataset. For each query image, the top (nearest) ten images returned by different hashing methods are shown. We use red rectangles to indicate the images that are not in the same class as the query image. That is to say, the images with red rectangles are wrongly returned results. It is easy to find that our LFH method exhibits minimal errors compared to other supervised hashing methods.

Figure 8: Example search (Hamming ranking) results on CIFAR-10, where red rectangles are used to indicate the images that are not in the same class as the query image, i.e., the wrongly returned results. (a) and (b) contain the same query images, which are duplicated for better demonstration. Other images are the returned results of (c) LFH; (d) KSH; (e) MLH; (f) SPLH.

4. CONCLUSION

Hashing has become a very effective technique for ANN search in massive datasets which are common in this big data era. In many datasets, it is not hard to collect some supervised information, such as the tag information in many social web sites, for part of the whole dataset. Hence, supervised hashing methods, which can outperform traditional unsupervised hashing methods, have become more and more important. In this paper, we propose a novel method, called LFH, for supervised hashing. A learning algorithm with convergence guarantee is proposed to learn the parameters of LFH. Moreover, to model large-scale problems, we propose a stochastic learning method which has linear time complexity. Experimental results on two large datasets with semantic labels show that our LFH method can achieve higher accuracy than other state-of-the-art methods with comparable or lower training cost.

5. ACKNOWLEDGMENTS

This work is supported by the NSFC (No. 61100125, 61272099, 61261160502), the 863 Program of China (No. 2012AA011003), Shanghai Excellent Academic Leaders Plan (No. 11XD1402900), Scientific Innovation Act of STCSM (No. 13511504200), and the Program for Changjiang Scholars and Innovative Research Team in University of China (IRT1158, PCSIRT).

6. REFERENCES

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *Proceedings of the Annual Symposium on Foundations of Computer Science*, pages 459–468, 2006.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[3] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y. Zheng. Nus-wide: A real-world web image database from national university of singapore. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, 2009.

[4] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Annual Symposium on Computational Geometry*, pages 253–262, 2004.

[5] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the International Conference on Very Large Data Bases*, pages 518–529, 1999.

[6] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 817–824, 2011.

[7] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the Annual ACM Symposium on Theory of Computing*, pages 604–613, 1998.

[8] W. Kong and W.-J. Li. Double-bit quantization for hashing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2012.

[9] W. Kong and W.-J. Li. Isotropic hashing. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 1655–1663, 2012.

[10] W. Kong, W.-J. Li, and M. Guo. Manhattan hashing for large-scale image retrieval. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 45–54, 2012.

[11] A. Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

[12] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 1042–1050, 2009.

[13] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2130–2137, 2009.

[14] B. Kulis, P. Jain, and K. Grauman. Fast similarity search for learned metrics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(12):2143–2157, 2009.

[15] K. Lange, D. R. Hunter, and I. Yang. Optimization transfer using surrogate objective functions. *Journal of Computational and Graphical Statistics*, 9(1):1–20, 2000.

[16] X. Li, G. Lin, C. Shen, A. van den Hengel, and A. R. Dick. Learning hash functions using column generation. In *Proceedings of the International Conference on Machine Learning*, pages 142–150, 2013.

[17] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 2074–2081, 2012.

[18] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *Proceedings of the International Conference on Machine Learning*, 2011.

[19] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *Proceedings of the International Conference on Machine Learning*, pages 353–360, 2011.

[20] M. Norouzi, D. J. Fleet, and R. Salakhutdinov. Hamming distance metric learning. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 1070–1078, 2012.

[21] M. Ou, P. Cui, F. Wang, J. Wang, W. Zhu, and S. Yang. Comparing apples to oranges: A scalable solution with heterogeneous hashing. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 230–238, 2013.

[22] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 1509–1517, 2009.

[23] M. Rastegari, J. Choi, S. Fakhraei, D. Hal, and L. S. Davis. Predictable dual-view hashing. In *Proceedings of the International Conference on Machine Learning*, pages 1328–1336, 2013.

[24] R. Salakhutdinov and G. E. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969–978, 2009.

[25] A. W. M. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain. Content-based image retrieval at the end of the early years. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(12):1349–1380, 2000.

[26] J. Song, Y. Yang, Y. Yang, Z. Huang, and H. T. Shen. Inter-media hashing for large-scale retrieval from heterogeneous data sources. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 785–796, 2013.

[27] B. Stein. Principles of hash-based text retrieval. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 527–534, 2007.

[28] C. Strecha, A. A. Bronstein, M. M. Bronstein, and P. Fua. Ldahash: Improved matching with smaller descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1):66–78, 2012.

[29] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

[30] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2008.

[31] F. Ture, T. Elsayed, and J. J. Lin. No free lunch: Brute force vs. locality-sensitive hashing for cross-lingual pairwise similarity. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 943–952, 2011.

[32] J. Wang, O. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 3424–3431, 2010.

[33] J. Wang, S. Kumar, and S.-F. Chang. Sequential projection learning for hashing with compact codes. In *Proceedings of the International Conference on Machine Learning*, pages 1127–1134, 2010.

[34] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Proceedings of the Annual Conference on Neural Information Processing Systems*, pages 1753–1760, 2008.

[35] F. Wu, Z. Yu, Y. Yang, S. Tang, Y. Zhang, and Y. Zhuang. Sparse multi-modal hashing. *IEEE Transactions on Multimedia*, 16(2):427–439, 2014.

[36] B. Xu, J. Bu, Y. Lin, C. Chen, X. He, and D. Cai. Harmonious hashing. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2013.

[37] D. Zhai, H. Chang, Y. Zhen, X. Liu, X. Chen, and W. Gao. Parametric local multimodal hashing for cross-view similarity search. In *Proceedings of the International Joint Conference on Artificial Intelligence*, 2013.

[38] D. Zhang and W.-J. Li. Large-scale supervised multimodal hashing with semantic correlation maximization. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2014.

[39] D. Zhang, F. Wang, and L. Si. Composite hashing with multiple information sources. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 225–234, 2011.

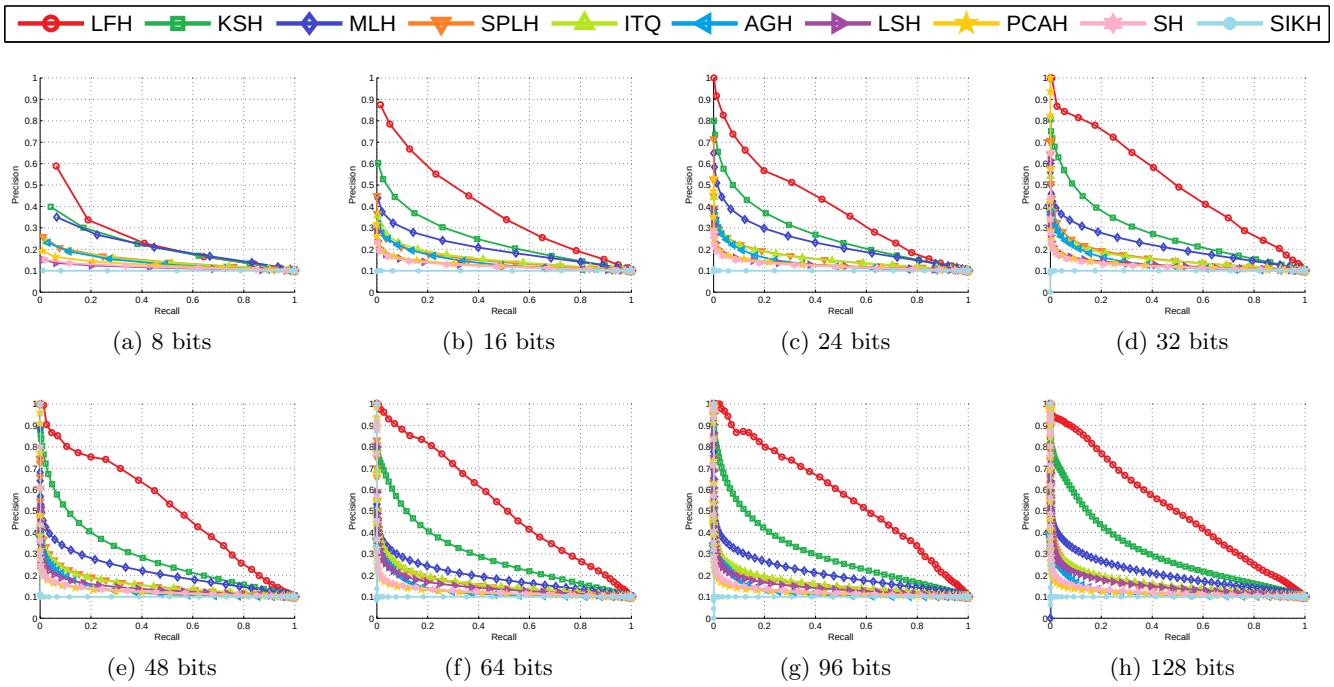
[40] D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 18–25, 2010.

[41] Q. Zhang, Y. Wu, Z. Ding, and X. Huang. Learning hash codes for efficient content reuse detection. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 405–414, 2012.

[42] Y. Zhen and D.-Y. Yeung. A probabilistic model for multimodal hash function learning. In *Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 940–948, 2012.

APPENDIX

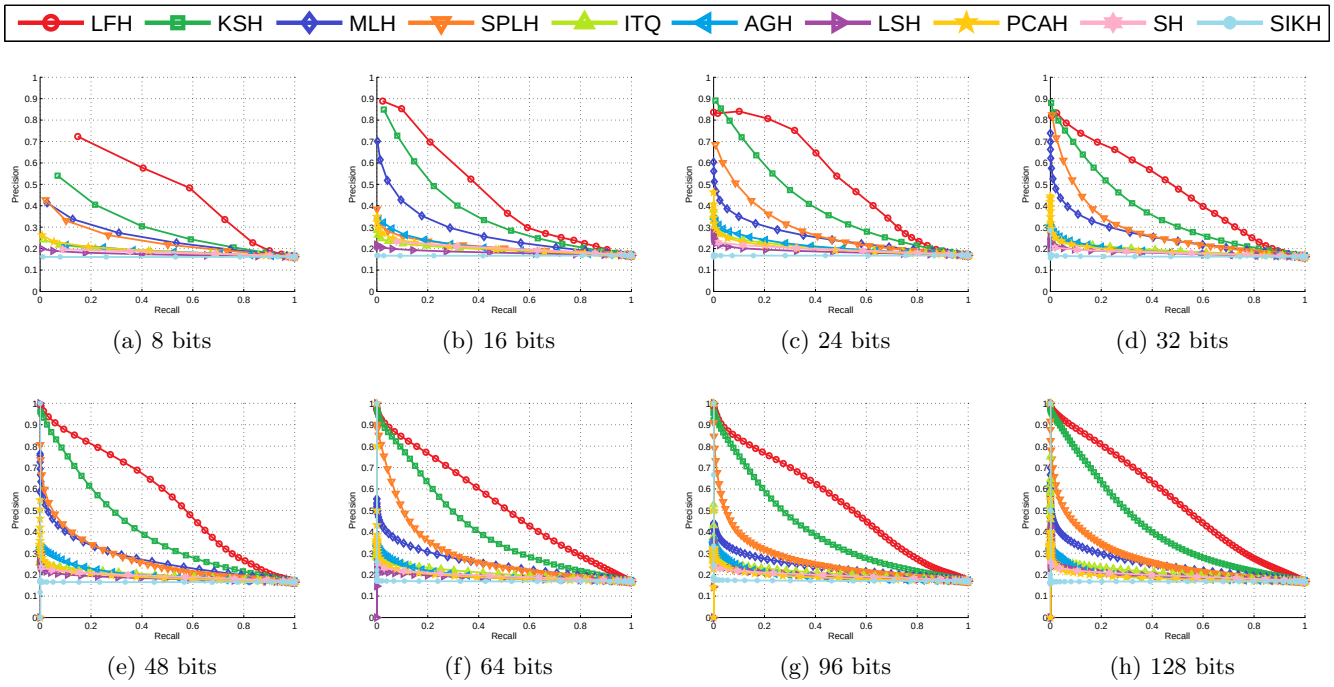
For a more extensive evaluation, in Figure 9 and Figure 10, we illustrate the precision-recall curves with different code lengths on the two datasets, CIFAR-10 and NUS-WIDE. Our LFH method shows clear superiority on almost all settings, followed by KSH, SPLH, and MLH, and then the other methods without using semantic information. The results are consistent with the MAP results given above.



(a) 8 bits (b) 16 bits (c) 24 bits (d) 32 bits

(e) 48 bits (f) 64 bits (g) 96 bits (h) 128 bits

Figure 9: Precision-recall curves on CIFAR-10.



(a) 8 bits (b) 16 bits (c) 24 bits (d) 32 bits

(e) 48 bits (f) 64 bits (g) 96 bits (h) 128 bits

Figure 10: Precision-recall curves on NUS-WIDE.