Bayesian Supervised Hashing

Zihao Hu, Junxuan Chen, Hongtao Lu*, and Tongzhen Zhang

Department of Computer Science and Engineering, Shanghai Jiao Tong University
{zihaohu, chenjunxuan, htlu}@sjtu.edu.cn, zhang-tz@cs.sjtu.edu.cn

Abstract

Among learning based hashing methods, supervised hashing seeks compact binary representation of the training data to preserve semantic similarities. Recent years have witnessed various problem formulations and optimization methods for supervised hashing. Most of them optimize a form of loss function with a regulization term, which can be viewed as a maximum a posterior (MAP) estimation of the hashing codes. However, these approaches are prone to overfitting unless hyperparameters are tuned carefully. To address this problem, we present a novel fully Bayesian treatment for supervised hashing problem, named Bayesian Supervised Hashing (BSH), in which hyperparameters are automatically tuned during optimization. Additionally, by utilizing automatic relevance determination (ARD), we can figure out relative discriminating ability of different hashing bits and select most informative bits among them. Experimental results on three real-world image datasets with semantic information show that BSH can achieve superior performance over state-of-the-art methods with comparable training time.

1. Introduction

Given a dataset, hashing methods map data points from the original feature space into a binary hashing code space with pairwise metric distances or semantic similarities preserved. Compact hashing codes can accelerate retrieval speed on the precondition of the accuracy. Hashing methods can be roughly divided into two main categories: data-independent methods and data-dependent methods.

The early exploration of hashing focuses on data-independent methods. The most popular method among them is Locality Sensitive Hashing (LSH) [4, 17, 1] which uses random projections to generate hash functions. Theoretically, with the increase of code length, the original distance is asymptotically preserved in Hamming space. Hence, long codes are usually used to obtain preferable retrieval precision. However, long codes would result in low recall rate with the usage of the hash lookup table . In practical applications, LSH utilizes multiple tables to ensure a reasonable recall. Both long codes and multiple tables will affect the efficiency of retrieval.

Data-dependent methods have been developed to learn more compact hashing codes by utilizing the training data. Due to shortcomings of data-independent methods, many researchers have been putting considerable efforts on designing data-dependent methods. Data-dependent methods can be further divided into two categories: unsupervised methods and supervised methods.

Unsupervised methods [19, 5, 10, 7, 20] just use the unlabeled data to generate binary codes, aiming at preserving the metric structure between data points. Representative method in this fashion is ITQ [5]. ITQ performs PCA to reduce the dimension of data points, then seeks an orthogonal matrix to minimize quantization loss between rotated data points and final binary codes. Compared with data-independent methods, unsupervised data-dependent methods need fewer bits to achieve comparable performance.

However, a variety of image searching applications prefer semantically similar neighbors. This leads to the formulation of supervised hashing. In contrast to unsupervised methods in which the training data is unlabeled, supervised hashing is designed to preserve semantic similarities by utilizing labeled training data. The representatives of supervised hashing methods include [18, 5, 14, 21, 16, 8, 13]. Among these methods, CCA-ITQ [5] and Supervised Discrete Hashing (SDH) [16] only utilize semantic labels of individual training points. CCA-ITQ first uses CCA to find projection directions to maximize correlations between feature and label vectors, and then applies standard ITQ to learn a rotation matrix . The learning objective of SDH is to generate hashing codes by minimizing linear classification loss function. Different from these two methods, most approaches are built upon pairwise semantic similarities.

*Corresponding author.

For instance, Kernel-based Supervised Hashing (KSH) [14] tries to solve a relaxed optimization problem to generate hashing codes with the pairwise similarity preserved. This idea is also used in other supervised methods.

While most of the hashing methods use discriminative models, latent factor models (LFH) proposes a generative model which assumes the pairwise semantic similarity is generated from two corresponding latent factors. By introducing some prior, the MAP estimation of latent factors is tractable. Latent factors are then rounded to acquire the final hashing codes. By introducing the generative model, LFH has strong power to infer underlying latent structures of semantic similarities and hence achieves favorable results.

However, providing a MAP estimation results in a practical challenge of choosing hyperparameters. Commonly used methods for selecting hyperparameters include grid search and random search on the validation set, which are inaccurate and time-consuming. The Bayesian approach can address this issue by tuning hyperparameters automatically during the optimization process . Hence, we propose a fully Bayesian probabilistic treatment, called Bayesian Supervised Hashing (BSH). The main contributions of this paper are outlined as follows:

1. We propose a novel supervised hashing method by utilizing the fully Bayesian treatment. Based on the variational inference, underlying latent factors can be inferred elegantly from semantic information while hyperparameters can be determined automatically.

2. During the learning process, automatic relevance determination (ARD) prior can determine the relative importance of different hashing bits. Thus, we can obtain shorter but more informative hashing bits based on learned longer codes without retraining, which makes BSH more flexible for real applications.

3. The proposed BSH is evaluated on three real-world datasets with semantic tags. Experimental results show that BSH outperforms other state-of-the-art methods. To tackle large-scale data, a linear-time variant is proposed. By the variant, training 64-bit hashing codes on the NUS-WIDE dataset costs less than 3 minutes.

The remainder of the paper is organized as follows: In Section 2 we introduce the notation used in this paper and review LFH briefly. Section 3 demonstrates the detail of our BSH model. Experimental results are presented in Section 4. Finally, we conclude the paper in Section 5.

2. Related work

2.1. Notation

Throughout this paper, vectors and matrices are denoted by boldface lower-case letters and boldface upper-case letters, respectively. Given a vector $\mathbf{x} = (x_1, \cdots, x_n)^T \in \mathbb{R}^n$, let $\mathrm{diag}(\mathbf{x})$ be an $n \times n$ diagonal matrix with the i-th diagonal element as x_i. For a matrix $\mathbf{A} = [a_{ij}] \in \mathbb{R}^{m \times n}$, a_{ij} represents the element at the i-th row and j-th column in $\mathbf{A}$, and $\mathbf{A}^T$ denotes the transpose of $\mathbf{A}$. $\mathbf{A}_{i\cdot}$ and $\mathbf{A}_{\cdot j}$ denote column vectors formed by the i-th row and the j-th column of $\mathbf{A}$, respectively. When $\mathbf{A}$ is square, we let $\mathbf{A}^{-1}$ be the inverse (if exist) of $\mathbf{A}$. $\|\cdot\|_F$ denotes the Frobenius norm of a matrix. $\mathbf{I}$ denotes the identity matrix of appropriate size. For probability distributions, $\mathcal{N}(\mathbf{x}|\boldsymbol{\mu}, \boldsymbol{\Sigma})$ denotes the multivariate normal distribution over a random vector $\mathbf{x}$ with mean vector $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, and $\mathcal{G}(\tau|\alpha, \beta)$ denotes the Gamma distribution over a random variable τ governed by parameters α and β.

2.2. Problem definition

Suppose we have n points $\{\mathbf{x}_i \in \mathbb{R}^d\}_{i=1}^n$ as the training data, where $\mathbf{x}_i$ is the i-th feature vector. These feature vectors can be collectively written in matrix form as $\mathbf{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_n]^T \in \mathbb{R}^{n \times d}$. In customary settings for supervised hashing [12, 21], the supervised information is given in terms of a semantic similarity matrix $\mathbf{S} \in \{0, 1\}^{n \times n}$, where $s_{ij} = 1$ means that point i and point j are semantically similar, while $s_{ij} = 0$ means that point i and point j are semantically dissimilar. In this paper, we assume that $\mathbf{S}$ is fully observed without missing entries. This assumption is reasonable because in most cases we can get the semantic similarity for arbitrary two points. The goal of supervised hashing is to learn a binary code matrix $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, \cdots, \mathbf{b}_n]^T = [b_{ij}] \in \{-1, 1\}^{n \times q}$, where $\mathbf{b}_i$ denotes the learned q-bit binary code for the i-th training point. Additionally, the semantic similarity in $\mathbf{S}$ should be preserved by the binary codes, in particular, if $s_{ij} = 1$, the Hamming distance between $\mathbf{b}_i$ and $\mathbf{b}_j$ should be low.

2.3. Brief review of LFH

Here we first briefly introduce the latent factor models proposed in [21]. Recall that minimizing the Hamming distance of two binary codes is equivalent to minimizing the inner product of them [14]. Hence, LFH assumes that the observed pairwise similarity is generated by the inner product of two corresponding hashing codes, the likelihood is defined as

$$p(s_{ij}|\mathbf{B}) = \begin{cases} \sigma(\frac{1}{2}\mathbf{b}_i^T\mathbf{b}_j), & s_{ij} = 1 \\ 1 - \sigma(\frac{1}{2}\mathbf{b}_i^T\mathbf{b}_j), & s_{ij} = 0 \end{cases}, \tag{1}$$

where $\mathbf{b}_i$ and $\mathbf{b}_j$ are binary codes of $\mathbf{x}_i$ and $\mathbf{x}_j$ respectively while σ denotes the sigmoid function.

The likelihood of the observed similarity matrix $\mathbf{S}$ can be written as

$$p(\mathbf{S}|\mathbf{B}) = \prod_{i=1}^{n}\prod_{j=1}^{n} p(s_{ij}|\mathbf{B}). \tag{2}$$

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

$$p(\mathbf{B}|\mathbf{S}) \propto p(\mathbf{S}|\mathbf{B})p(\mathbf{B}). \tag{3}$$

Direct MAP estimation of $\mathbf{B}$ is intractable [19], therefore, $\mathbf{B}$ is computed through two stages. In the first stage, $\mathbf{B}$ is relaxed to be a matrix $\mathbf{U} = [u_{ij}] \in \mathbb{R}^{n\times q}$ and the optimal $\mathbf{U}$ is learned. The i-th row of $\mathbf{U}$ is called the i-th latent factor. Then in the second stage, $\mathbf{U}$ is rounded to acquire the binary code matrix $\mathbf{B}$.

During the learning process, each row of $\mathbf{U}$ is optimized at a time with other rows fixed. For a certain row, a lower bound is constructed by the second order Taylor expansion of log posterior $L = \log p(\mathbf{U}|\mathbf{S})$, and then Newton's method is utilized to update the latent factor of the row.

3. Bayesian supervised hashing

3.1. Model formulation

Our proposed method adopts similar likelihood as in LFH [21] with slight modification:

$$p(s_{ij}|\mathbf{U}) = \begin{cases} \sigma(\mathbf{U}_{i\cdot}^T\mathbf{U}_{j\cdot}), & s_{ij} = 1 \\ 1 - \sigma(\mathbf{U}_{i\cdot}^T\mathbf{U}_{j\cdot}), & s_{ij} = 0 \end{cases}. \tag{4}$$

The prior over latent factor matrix $\mathbf{U}$ takes the form

$$p(\mathbf{U}|\boldsymbol{\gamma}) = \prod_{i=1}^{q}\mathcal{N}(\mathbf{U}_{\cdot i}|\mathbf{0}, \gamma_i^{-1}\mathbf{I}) = \prod_{i=1}^{n}\mathcal{N}(\mathbf{U}_{i\cdot}|\mathbf{0}, \boldsymbol{\Gamma}^{-1}) \tag{5}$$

where $\boldsymbol{\Gamma} = \mathrm{diag}(\boldsymbol{\gamma}) \in \mathbb{R}^{q\times q}$ is a diagonal matrix with the i-th diagonal element as γ_i. Instead of a single shared hyperparameter, we introduce a separate hyperparameter γ_i for each $\mathbf{U}_{\cdot i}$. This formulation therefore is the analogue of automatic relevance determination (ARD) successfully utilized for sparse Bayesian learning. During inference, if some hyperparameter γ_i assumes large value, then the posterior distribution over $\mathbf{U}_{\cdot i}$ will be concentrated at $\mathbf{0}$, which denotes that the i-th hashing bit is relatively inessential. Conversely, small value of a hyperparameter is a clear proof of its discrimination.

To complete the fully Bayesian treatment, we consider a conjugate hyperprior over $\boldsymbol{\gamma}$ given by the Gamma distribution

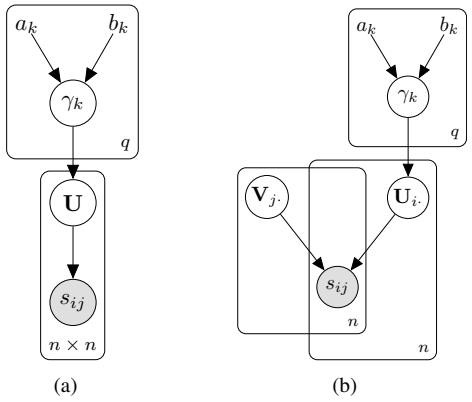
$$p(\boldsymbol{\gamma}|\mathbf{a}, \mathbf{b}) = \prod_{k=1}^{q}\mathcal{G}(\gamma_k|a_k, b_k) \tag{6}$$

governed by constant vectors $\mathbf{a} = [a_1, a_2, \cdots, a_q]^T$ and $\mathbf{b} = [b_1, b_2, \cdots, b_q]^T$.

The joint distribution, therefore, is expressed as

$$p(\mathbf{S}, \mathbf{U}, \boldsymbol{\gamma}) = p(\mathbf{S}|\mathbf{U})p(\mathbf{U}|\boldsymbol{\gamma})p(\boldsymbol{\gamma}|\mathbf{a}, \mathbf{b}). \tag{7}$$

The corresponding graphical model is shown in Figure 1(a).



(a) (b)

Figure 1. Probabilistic graphical models for the joint distribution before and after introducing $\mathbf{V}$.

3.2. Bayesian inference

Exact Bayesian inference for all unknowns $\mathbf{U}$ and $\boldsymbol{\gamma}$ using the joint distribution in Equation (7) is intractable, since the marginal likelihood $p(\mathbf{S})$ cannot be computed analytically. Therefore, approximation methods is utilized. Note that the form of likelihood function in Equation (4) is similar to that of logistic regression:

$$p(y|\mathbf{w}) = \begin{cases} \sigma(\mathbf{w}^T\mathbf{x}), & y = 1 \\ 1 - \sigma(\mathbf{w}^T\mathbf{x}), & y = 0 \end{cases}. \tag{8}$$

Hence, we can consider using similar treatment as for Bayesian logistic regression. The sigmoid function has no conjugate prior but has a tight bound [2]

$$\sigma(x) \geq \sigma(\xi)\exp\{(x-\xi)/2 + \lambda(\xi)(x^2 - \xi^2)\}, \tag{9}$$

where ξ is the variational parameter and

$$\lambda(\xi) = -\frac{1}{4\xi}\tanh\left(\frac{\xi}{2}\right). \tag{10}$$

Adopting this bound, the posterior over $\mathbf{w}$ will be a Gaussian. However, in Equation (4), $\mathbf{U}_{i\cdot}$ and $\mathbf{U}_{j\cdot}$ are both latent variables, so the posterior over $\mathbf{U}_{i\cdot}$ will be a quartic exponential distribution considering s_{ii}, which is hard to marginalize.

The authors of [15] show that shorter and more accurate hashing codes can be acquired by introducing asymmetry carefully on the formulation of supervised hashing. Inspired by this, we assume that the similarity matrix $\mathbf{S}$ is generated from two different matrices $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{n\times q}$

$$p(s_{ij}|\mathbf{U}) = \begin{cases} \sigma(\mathbf{U}_{i\cdot}^T\mathbf{V}_{j\cdot}), & s_{ij} = 1 \\ 1 - \sigma(\mathbf{U}_{i\cdot}^T\mathbf{V}_{j\cdot}), & s_{ij} = 0 \end{cases}. \tag{11}$$

$\mathbf{V}$ is introduced to break the symmery in Equation (4) and we do not place prior on $\mathbf{V}$. This formulation also results in

that the posterior over $\mathbf{U}_{i\cdot}$ becomes a Gaussian. Figure 1(b) shows the graphical model after introducing $\mathbf{V}$.

At the beginning of an iteration, we assume that $\mathbf{V}_{i\cdot}$ has the same distribution as $\mathbf{U}_{i\cdot}$ to preserve a similar structure as in Equation (4). Since the distribution over $\mathbf{U}_{i\cdot}$ is a Gaussian, we update mean vector and covariance matrix of $\mathbf{V}_{i\cdot}$

$$\mathbb{E}[\mathbf{V}_{i\cdot}] = \mathbb{E}[\mathbf{U}_{i\cdot}], \tag{12}$$

$$\mathbb{D}[\mathbf{V}_{i\cdot}] = \mathbb{D}[\mathbf{U}_{i\cdot}]. \tag{13}$$

During the optimization process, we update $\mathbf{U}$ and $\boldsymbol{\gamma}$ with the distribution of $\mathbf{V}$ fixed. Introducing $\mathbf{V}$ may break the integrity of the Bayesian approach, but the experimental performance is still satisfactable as we shall see later.

After replacing the likelihood (4) with (11), we can adopt a variational inference to work out this problem, which is based on maximizing a lower bound $\mathcal{L}(q)$ on the marginal data log-likelihood

$$\begin{aligned}\log p(\mathbf{S}) &= \log \iint p(\mathbf{S}|\mathbf{U})p(\mathbf{U}|\boldsymbol{\gamma})p(\boldsymbol{\gamma})\mathrm{d}\mathbf{U}\mathrm{d}\boldsymbol{\gamma} \\ &\geq \mathcal{L}(q),\end{aligned} \tag{14}$$

where

$$\mathcal{L}(q) = \iint q(\mathbf{U},\boldsymbol{\gamma}) \log \frac{p(\mathbf{S}|\mathbf{U})p(\mathbf{U}|\boldsymbol{\gamma})p(\boldsymbol{\gamma})}{q(\mathbf{U},\boldsymbol{\gamma})}\mathrm{d}\mathbf{U}\mathrm{d}\boldsymbol{\gamma} \tag{15}$$

and $q(\mathbf{U},\boldsymbol{\gamma})$ is an approximate posterior distribution. We assume $q(\mathbf{U},\boldsymbol{\gamma})$ has the following factorized form:

$$q(\mathbf{U},\boldsymbol{\gamma}) = \prod_{i=1}^{n} q(\mathbf{U}_{i\cdot}) \prod_{k=1}^{q} q(\gamma_k). \tag{16}$$

Applying the bound in Equations (9) and (10) to Equation (11), the likelihood for s_{ij} is lower-bounded by

$$\begin{aligned}p(s_{ij}|\mathbf{U}) &= (\sigma(\theta_{ij}))^{s_{ij}}(1-\sigma(\theta_{ij}))^{1-s_{ij}} \\ &= \sigma(-\theta_{ij})e^{s_{ij}\theta_{ij}} \\ &\geq \exp\{(s_{ij}-\frac{1}{2})\theta_{ij} + \lambda(\xi_{ij})\theta_{ij}^2 \\ &\quad + \log\sigma(\xi_{ij}) - \frac{1}{2}\xi_{ij} - \lambda(\xi_{ij})\xi_{ij}^2\},\end{aligned} \tag{17}$$

where $\theta_{ij} = \mathbf{U}_{i\cdot}^T\mathbf{V}_{j\cdot}$. Hence, the bound on $p(\mathbf{S}|\mathbf{U})$ is

$$\begin{aligned}p(\mathbf{S}|\mathbf{U}) &\geq h(\mathbf{U},\boldsymbol{\xi}) \\ &= \prod_{i=1}^{n}\prod_{j=1}^{n} \exp\{(s_{ij}-\frac{1}{2})\theta_{ij} + \lambda(\xi_{ij})\theta_{ij}^2 \\ &\quad + \log\sigma(\xi_{ij}) - \frac{1}{2}\xi_{ij} - \lambda(\xi_{ij})\xi_{ij}^2\},\end{aligned} \tag{18}$$

where $\boldsymbol{\xi}$ denotes the set $\{\xi_{ij}\}$ of variational parameters. This results in a new lower bound

$$\widetilde{L}(q,\boldsymbol{\xi}) = \iint q(\mathbf{U},\boldsymbol{\gamma}) \log \frac{h(\mathbf{U},\boldsymbol{\xi})p(\mathbf{U}|\boldsymbol{\gamma})p(\boldsymbol{\gamma})}{q(\mathbf{U},\boldsymbol{\gamma})}\mathrm{d}\mathbf{U}\mathrm{d}\boldsymbol{\gamma}. \tag{19}$$

Since $\log p(\mathbf{S}) \geq \mathcal{L}(q) \geq \widetilde{L}(q,\boldsymbol{\xi})$ always holds, we can optimize the lower bound $\widetilde{L}(q,\boldsymbol{\xi})$ with respect to $q(\mathbf{U},\boldsymbol{\gamma})$ and $\boldsymbol{\xi}$ to maximize $\log p(\mathbf{S})$.

Following standard variational treatment [2], the approximate posterior for $\mathbf{U}$ is given by

$$\begin{aligned}\log q(\mathbf{U}) &= \log h(\mathbf{U},\boldsymbol{\xi}) + \mathbb{E}_{\boldsymbol{\gamma}}[\log p(\mathbf{U}|\boldsymbol{\gamma})] + \text{const} \\ &= \sum_{i=1}^{n} \log \mathcal{N}(\mathbf{U}_{i\cdot}|\boldsymbol{\mu}_i^\star, \boldsymbol{\Sigma}_i^\star),\end{aligned} \tag{20}$$

where

$$\boldsymbol{\mu}_i^\star = \boldsymbol{\Sigma}_i^\star \sum_{j=1}^{n} \mathbb{E}[\mathbf{V}_{j\cdot}](s_{ij}-\frac{1}{2}), \tag{21}$$

$$\boldsymbol{\Sigma}_i^\star = \left\{\mathbb{E}[\boldsymbol{\Gamma}] - 2\sum_{j=1}^{n}\lambda(\xi_{ij})\mathbb{E}[\mathbf{V}_{j\cdot}\mathbf{V}_{j\cdot}^T]\right\}^{-1}. \tag{22}$$

To find the local variational parameter ξ_{ij} that maximizes $\widetilde{L}(q,\boldsymbol{\xi})$, its derivative with respect to ξ_{ij} is set to zero, and we obtain the re-estimation equation

$$(\xi_{ij}^\star)^2 = \mathbb{E}[\mathbf{U}_{i\cdot}^T\mathbf{U}_{i\cdot}]\mathbb{E}[\mathbf{V}_{j\cdot}^T\mathbf{V}_{j\cdot}]. \tag{23}$$

Similarly, the optimal solution for the variational posterior $q^\star(\boldsymbol{\gamma})$ is obtained from

$$\begin{aligned}\log q(\boldsymbol{\gamma}) &= \mathbb{E}_{\mathbf{U}}[\log p(\mathbf{U}|\boldsymbol{\gamma})] + \log p(\boldsymbol{\gamma}) + \text{const} \\ &= \sum_{k=1}^{q} \log \mathcal{G}(\gamma_k|a_k^\star, b_k^\star),\end{aligned} \tag{24}$$

with parameters

$$a_k^\star = a_k + \frac{n}{2}, \tag{25}$$

$$b_k^\star = b_k + \frac{1}{2}\mathbb{E}[\mathbf{U}_{\cdot k}^T\mathbf{U}_{\cdot k}]. \tag{26}$$

The required moments are given by

$$\mathbb{E}[\boldsymbol{\Gamma}] = \mathrm{diag}\left(\frac{a_1^\star}{b_1^\star}, \frac{a_2^\star}{b_2^\star}, \cdots, \frac{a_q^\star}{b_q^\star}\right), \tag{27}$$

$$\mathbb{E}[\mathbf{U}_{\cdot k}^T\mathbf{U}_{\cdot k}] = \sum_{i=1}^{n}\left((\boldsymbol{\mu}_i^\star)_k^2 + (\boldsymbol{\Sigma}_i^\star)_{kk}\right). \tag{28}$$

Algorithm 1 shows the pseudocode of the learning process.

3.3. Rounding

After the optimal $\mathbf{U}$ is learned, we can obtain the final binary codes using some rounding techniques. Here we simply round $\mathbf{U}$ according to the mean value of each column

$$b_{ij} = \begin{cases}1, & u_{ij} > \text{mean}(\mathbf{U}_{\cdot j}) \\ -1, & \text{otherwise}\end{cases}. \tag{29}$$

Algorithm 1: Variational Bayesian hashing learning

Input : $\mathbf{X} \in \mathbb{R}^{n\times d}, \mathbf{S} \in \{0,1\}^{n\times n}, q, \tau \in \mathbb{N}^+$.
Output: $\mathbf{U} \in \mathbb{R}^{n\times q}$, which will be rounded to obtain the binary code matrix $\mathbf{B} \in \{0,1\}^{n\times q}$.

Sample mean $\boldsymbol{\mu}_i \sim \mathcal{N}(\mathbf{0},\ \mathbf{I})\ , i = 1, 2, \cdots, n$.
Set covariance matrix $\boldsymbol{\Sigma}_i = \mathbf{I}, i = 1, 2, \cdots, n$.
Set variational parameters $\xi_{ij} = 0$.
Set $a_k = b_k = 2 \times 10^{-3}, k = 1, \cdots, q$.
for $t \leftarrow 1$ *to* τ **do**
  Update $\mathbf{V}$ by Equations (12) and (13).
  Update $\mathbf{U}$ by Equations (21) and (22).
  Update all the ξ_{ij} and $\lambda(\xi_{ij})$ by (23) and (10).
  Update hyperparameters $\mathbf{a}$ and $\mathbf{b}$ according to Equations (25) and (26).
end
return $\mathbf{U} = [\boldsymbol{\mu}_1^\star, \boldsymbol{\mu}_2^\star, \cdots, \boldsymbol{\mu}_n^\star]^T$.

3.4. Out-of-sample extension

The procedure described in Section 3.2 creates hashing codes only for the training samples. For query points, we construct hashing codes using linear regression for simplicity. To achieve this, we seek a linear mapping $\mathbf{W} \in \mathbb{R}^{d\times q}$ to transform data matrix $\mathbf{X}$ into $\mathbf{U}$. The squared loss function with regularization term is

$$L = \|\mathbf{U} - \mathbf{X}\mathbf{W}\|_F^2 + \lambda\|\mathbf{W}\|_F^2, \tag{30}$$

and $\mathbf{W}$ has closed form optimal solution as

$$\mathbf{W} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I})^{-1}\mathbf{X}^T\mathbf{U}. \tag{31}$$

3.5. Stochastic Learning

It is obvious that both time complexity and storage consumption are $\mathcal{O}(n^2)$ if all the supervised information in S is used for training, which is impracticable for large-scale data. Hence, we randomly sample m columns in $\mathbf{S}$ to form semantic information matrix $\widetilde{S}$ for training, that is, $\widetilde{S} = [\mathbf{S}_{\cdot i_1}, \cdots, \mathbf{S}_{\cdot i_m}]$. Compared to learning from the whole $\mathbf{S}$, we only need to change the index set of j from $\{1, \cdots, n\}$ to $\mathcal{I} = \{i_1, \cdots, i_m\}$. For instance, Equations (21) and (22) become

$$\boldsymbol{\mu}_i^\star = \boldsymbol{\Sigma}_i^\star \sum_{j\in\mathcal{I}} \mathbb{E}[\mathbf{V}_{j\cdot}](s_{ij} - \frac{1}{2}), \tag{32}$$

$$\boldsymbol{\Sigma}_i^\star = \left\{ \mathbb{E}[\boldsymbol{\Gamma}] - 2\sum_{j\in\mathcal{I}} \lambda(\xi_{ij})\mathbb{E}[\mathbf{V}_{j\cdot}\mathbf{V}_{j\cdot}^T] \right\}^{-1}. \tag{33}$$

3.6. Complexity analysis

For stochastic learning, $\mathbf{V} \in \mathbb{R}^{m\times q}$. At the beginning of each iteration, we update the sufficient statistic of $\mathbf{V}$ at the cost of $\mathcal{O}(mq + mq^2)$. Then it takes $\mathcal{O}(mq^2 + q^3)$ and $\mathcal{O}(mq + q^2)$ to compute the covariance matrix and mean vector of $\mathbf{U}_{i\cdot}$, respectively. Hence, the total time of updating $\mathbf{U}$ is $\mathcal{O}((mq + q^2 + mq^2 + q^3)n)$. Re-estimating all the variational parameters ξ_{ij} and $\lambda(\xi_{ij})$ costs $\mathcal{O}(mnq^2 + mn)$ time. For hyperparameters $\mathbf{a}$ and $\mathbf{b}$, we can update them at the cost of $\mathcal{O}(q + qn)$. Thus, the total time of an iteration is $\mathcal{O}((2mq^2 + q^3)n)$, which is linear to n with the typical assumption $n \gg \max\{m, q\}$. However, if we choose the whole $\mathbf{S}$ for learning, the time cost of an iteration will be $\mathcal{O}(n^2q^2)$, which is unacceptable when n grows large.

Besides that, the time for rounding is $\mathcal{O}(nq)$ and the time to compute $\mathbf{W}$ for out-of-sample extension is $O(nqd + nd^2 + d^3)$. With predetermined $\mathbf{W}$, the out-of-sample extension for a query can be achieved in $\mathcal{O}(qd)$.

For storage cost, the mean vectors and covariance matrices of $\mathbf{U}, \mathbf{V}$ require $\mathcal{O}((m + n)(q + q^2))$ in total, while variational parameters ξ_{ij} and hyperparameters $\mathbf{a}, \mathbf{b}$ occupy $\mathcal{O}(mn)$ and $\mathcal{O}(q)$, respectively. Therefore, the total storage cost is $\mathcal{O}((q^2 + m)n)$, which is also linear to n.

Consequently, since q is usually very small, *e.g.* less than 64 and m is typically set as 1000, we can say that both time complexity and storage cost of our proposed method are linear to the number of training samples, which make BSH easily scalable to very large datasets.

4. Experiment

4.1. Datasets

Three image datasets with semantic tags are used to evaluate our proposed method and other baselines. They are IAPRTC12[1] [6], ESPGAME[2], and NUS-WIDE[3] [3]. All of them have been widely used for supervised hashing methods evaluation [9, 11, 13].

The IAPRTC12 benchmark contains 19627 natural images with 291 tags while the ESPGAME dataset consists of 20770 images and 268 keywords. Each image may have multiple tags (keywords). Each image in both datasets is represented by a 512-dimensional GIST feature vector.

The NUS-WIDE dataset includes 269648 images collected from Flickr with 81 tags. We use 500 dimensional bag-of-words vectors for the experiments. It is worth noting that maintainers of NUS-WIDE have released tags crawled from websites with simple processing and ground truth annotated manually. In the following sections we use NUS-WIDE-GND and NUS-WIDE-TAG to denote the feature vectors of NUS-WIDE with ground truth and tags, respectively. NUS-WIDE-GND and NUS-WIDE-TAG associate each sample with 1.87 and 0.86 tags on average, respectively. Therefore, learning semantic information from NUS-

[1] http://www.imageclef.org/photodata
[2] http://www.hunch.net/ jl/
[3] http://lms.comp.nus.edu.sg/research/NUS-WIDE.htm

Method	ESPGAME				IAPRTC12			
	8 bits	16 bits	32 bits	64 bits	8 bits	16 bits	32 bits	64 bits
CCA-ITQ	0.2763	0.2809	0.2787	0.2811	0.3730	0.3756	0.3771	0.3804
LFH	0.3212	0.3356	0.3584	0.3611	0.4175	0.4401	0.4705	0.4658
SDH	0.2388	0.2422	0.2416	0.2546	0.3075	0.3158	0.3158	0.3344
COSDISH	0.2970	0.3149	0.3318	0.3458	0.3995	0.4168	0.4495	0.4649
NSH	0.2951	0.3013	0.3088	0.3134	0.3992	0.4144	0.4183	0.4253
BSH	**0.3349**	**0.3472**	**0.3593**	**0.3684**	**0.4451**	**0.4612**	**0.4784**	**0.4889**

Table 1. Experimental performance on ESPGAME and IAPRTC12 datasets in terms of mAP. Best results are in bold.

Method	NUS-WIDE-GND				NUS-WIDE-TAG			
	8 bits	16 bits	32 bits	64 bits	8 bits	16 bits	32 bits	64 bits
CCA-ITQ	0.2902	0.3023	0.3052	0.3163	0.0626	0.0665	0.0671	0.0678
LFH	0.4369	0.4745	0.5110	**0.5359**	0.0437	0.1241	0.1779	0.2038
SDH	0.2138	0.2161	0.2137	0.2143	0.0401	0.0406	0.0405	0.0406
COSDISH	0.4376	0.4774	0.5030	0.5189	0.1041	0.1412	0.1593	0.1848
NSH	0.3287	0.3407	0.3436	0.3508	0.0651	0.0686	0.0736	0.0746
BSH	**0.4751**	**0.4835**	**0.5146**	0.5256	**0.1635**	**0.1868**	**0.1955**	**0.2072**

Table 2. Experimental performance on the NUS-WIDE dataset in terms of mAP. Best results are in bold.

WIDE-TAG is more challenging. Both NUS-WIDE-GND and NUS-WIDE-TAG are evaluated in our experiments.

For these three datasets, two images are considered semantically similar if they share at least one common tag, otherwise, they are treated semantically dissimilar.

4.2. Experimental settings

As in [21, 8], for all datasets we randomly choose 1000 samples as the query set, with rest points as the training set. Note that our method does not need to set up a validation set since hyperparameters for the training set can be automatically determined during the learning phase. For out-of-sample extension, we simply set the hyperparameter λ according to [21]. In the preprocessing phase, we perform normalization on features to make each dimension have zero mean and equal variance. All our experiments are conducted on a workstation with 24 Intel Xeon CPU cores and 48 GB RAM, and all the results are the average value of 10 independent partitions.

Since existing studies [21, 13] have shown that supervised methods outperform unsupervised approaches, we only compare our method with several representative supervised hashing methods, including CCA-ITQ [5], LFH [21], SDH [16], COSDISH [8] and NSH [13]. All the baselines are implemented by the source code provided by the corresponding authors. All the hyperparameters and initialization strategies are the same as those suggested by the authors of these methods. For our method, we set $\tau = 2$ and $m = 1000$ in all experiments.

4.3. Hamming ranking performance

We perform Hamming ranking utilizing the generated binary codes on ESPGAME, IAPRTC12 and NUS-WIDE datasets. For each query, all the points in the training set are ranked according to the Hamming distance between their binary codes and the query's. The mean average precision (mAP) is reported to evaluate the performance of different supervised hashing methods.

Tables 1 and 2 show the mAP of our method and other baselines over these three datasets. By comparing BSH with CCA-ITQ, LFH, SDH, COSDISH and NSH, we can find that BSH outperforms other baselines in most cases. This is due to that other methods can be viewed as a MAP estimation. Their manually tuned hyperparameters are strongly depended on the validation set, and hence may be unsuitable for the test set. Our method, however, can tune hyperparameters automatically simply by using the training set. Therefore, BSH has better generalization ability. Additionally, we can think the code length q as the hyperparameter controlling the model complexity of a hashing method. With small q (≤ 32), other methods are not adapted to the rigid model and give very poor results. BSH, however, still yields desirable results by tuning the model complexity automatically. For instance, by using 8 bits, BSH can yield comparable performance with other baselines using 16 bits. Recall that short codes allow for very fast Hamming distance calculations. More importantly, if we would like to store the database in a hash table allowing immediate lookup, the size of the hash table is exponential in the code length. Thus it limits the length of binary codes to at most 64 bits, so the

retrieval performance of short codes is crucial in practical big-data applications.

It is also worth noting that for noisy and sparsely tagged datasets such as NUS-WIDE-TAG (0.86 tags per sample), pairwise similarity is rare. The Bayesian approach, instead of using point estimation, averages multiple models with respect to the posterior distribution learned from data, and hence can obtain reasonable performance with small amount of pairwise similarities. In summary, BSH can achieve better accuracy than state-of-the-art methods and is robust to noise.

Figure 3 shows some retrieval results on the NUS-WIDE dataset. This dataset is obviously challenging but the majority of retrieved images seem to be semantically relevant.

4.4. Computational cost

Table 3 shows the training time in seconds on NUS-WIDE-GND with various code lengths utilizing BSH and other baselines. We can see that, all these methods can handle the whole training set of NUS-WIDE (≈270K samples). Among them, only CCA-ITQ is faster than our method. However, as shown in Tables 1 and 2, BSH outperforms CCA-ITQ by a very large margin. Considering the retrieval precision and the training time complexity, BSH is the most suitable method for supervised hashing.

Method	8 bits	16 bits	32 bits	64 bits
CCA-ITQ	8.299	10.930	16.411	29.563
LFH	25.828	34.557	51.934	89.469
SDH	72.380	114.516	87.152	170.817
COSDISH	26.195	57.287	172.352	621.564
NSH	52.224	60.235	76.648	106.320
BSH	12.811	17.829	40.062	140.348

Table 3. Training time on NUS-WIDE-GND with various number of bits in seconds.

4.5. Hashing bits selection

By utilizing ARD in Equation (5), we can determine the relative discriminating ability of each learned hashing bit. To validate this argument, we learn a 64-bit length hash function and simply select k-bit codes according to the indexes of the k smallest ARD hyperparameters in $\boldsymbol{\gamma}$. The mAP of this method is compared with the mAP of the k-bit binary codes directly learning using BSH. We choose $k = 4, 8, \cdots, 64$ in actual experiments.

As shown in Figure 2, we can see that the mAP curve of selected bits (the blue curve) is smooth, concave and rather close to the red curve on three datasets consistently. The concavity of the blue curve illustrates that the hashing bits being selected earlier indeed are more discriminating. And empirically, selecting 32-bit codes from 64-bit learned

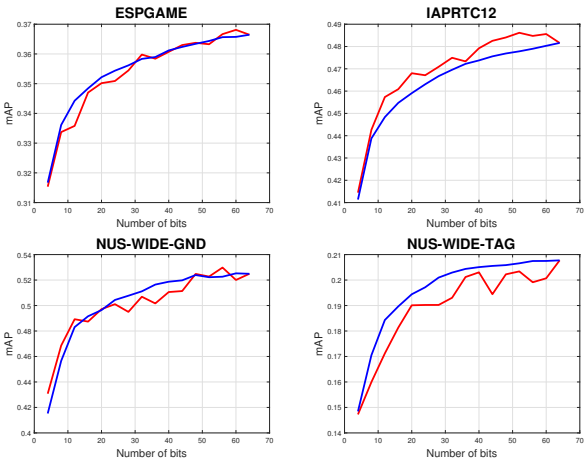


Figure 2. The red curve denotes the mAP of the k-bit hashing codes learned directly using BSH, while the blue curve denotes the mAP of k-bit hashing codes selected from already learned 64 bits according to ARD. $k = 4, 8, \cdots, 64$.

hashing codes results in 1% declination of mAP. Due to the scalability of BSH as shown in Section 4.4, we can learn longer hashing codes, and retain shorter but more discriminating codes for long term storage, which makes BSH more flexible for real applications.

5. Conclusion

In this paper we have proposed a novel supervised hashing method called BSH. By adopting a fully Bayesian treatment based on the variational inference, hyperparameters can be automatically tuned simply using the training data. By adopting a powerful Bayesian sparse learning tool called ARD, we can determine the relative discriminating ability of different bits, and select the most informative bits among them, which makes our method flexible for real applications. Experiments on several image datasets show that BSH outperforms other state-of-the-art methods.

6. Acknowledgements

This paper is supported by NSFC (No.61272247, 61533012 and 61472075), the 863 High-Tech Program (SS2015AA020501), and the Shanghai Science and Technology Committee (16JC1402800, 15JC1400103).

References

[1] A. Andoni, P. Indyk, T. Laarhoven, I. Razenshteyn, and L. Schmidt. Practical and optimal lsh for angular distance. In *Advances in Neural Information Processing Systems*, pages 1225–1233, 2015.

[2] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

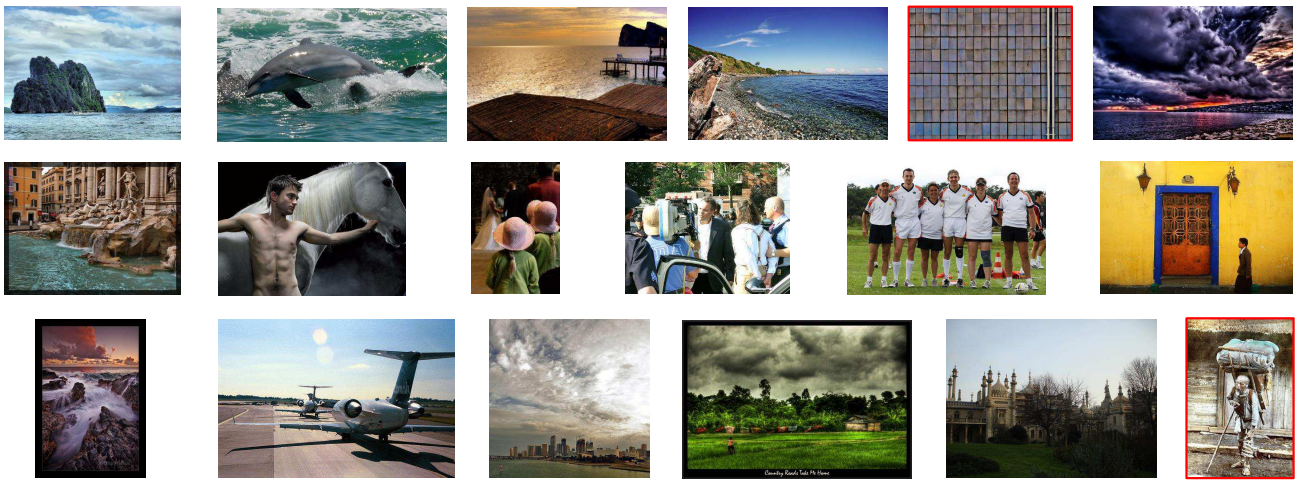


Figure 3. Some retrieval examples of BSH on the NUS-WIDE dataset. The first column shows query images and the rest are top 5 returned images in the dataset. False predictions are marked by red boxes.

[3] T.-S. Chua, J. Tang, R. Hong, H. Li, Z. Luo, and Y.-T. Zheng. Nus-wide: A real-world web image database from national university of singapore. In *Proc. of ACM Conf. on Image and Video Retrieval (CIVR'09)*, Santorini, Greece., July 8-10, 2009.

[4] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, pages 518–529, 1999.

[5] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 817–824. IEEE, 2011.

[6] M. Grubinger, P. Clough, H. Müller, and T. Deselaers. The iapr tc-12 benchmark: A new evaluation resource for visual information systems. In *OntoImage 2006 Workshop on Language Resources for Content-based Image Retrieval during LREC 2006 Final Programme*.

[7] G. Irie, Z. Li, X.-M. Wu, and S.-F. Chang. Locally linear hashing for extracting non-linear manifolds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2115–2122, 2014.

[8] W.-C. Kang, W.-J. Li, and Z.-H. Zhou. Column sampling based discrete supervised hashing. In *AAAI*, 2016.

[9] R. Kiros and C. Szepesvári. Deep representations and codes for image auto-annotation. In *Advances in Neural Information Processing Systems*, pages 908–916, 2012.

[10] W. Kong and W.-J. Li. Isotropic hashing. In *Advances in Neural Information Processing Systems*, pages 1646–1654, 2012.

[11] G. Lin, C. Shen, Q. Shi, A. van den Hengel, and D. Suter. Fast supervised hashing with decision trees for high-dimensional data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1963–1970, 2014.

[12] G. Lin, C. Shen, D. Suter, and A. van den Hengel. A general two-step approach to learning-based hashing. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2552–2559, 2013.

[13] Q. Liu and H. Lu. Natural supervised hashing. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, pages 1788–1794, 2016.

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

[15] B. Neyshabur, N. Srebro, R. R. Salakhutdinov, Y. Makarychev, and P. Yadollahpour. The power of asymmetry in binary hashing. In *Advances in Neural Information Processing Systems*, pages 2823–2831, 2013.

[16] F. Shen, C. Shen, W. Liu, and H. Tao Shen. Supervised discrete hashing. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 37–45, 2015.

[17] A. Shrivastava and P. Li. Asymmetric lsh (alsh) for sublinear time maximum inner product search (mips). In *Advances in Neural Information Processing Systems*, pages 2321–2329, 2014.

[18] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3424–3431. IEEE, 2010.

[19] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *Advances in neural information processing systems*, pages 1753–1760, 2009.

[20] F. X. Yu, S. Kumar, Y. Gong, and S.-F. Chang. Circulant binary embedding. In *International conference on machine learning*, volume 6, page 7, 2014.

[21] P. Zhang, W. Zhang, W.-J. Li, and M. Guo. Supervised hashing with latent factor models. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*, pages 173–182. ACM, 2014.