Noname manuscript No.
(will be inserted by the editor)

Maximum Margin Hashing with Supervised Information

**Haichuan Yang, Xiao Bai, Yanzhen Liu,
Yanyang Wang, Lu Bai, Jun Zhou,
Wenzhong Tang**

Received: date / Accepted: date

Abstract Binary code is a kind of special representation of data. With the binary format, hashing framework can be built and a large amount of data can be indexed to achieve fast research and retrieval. Many supervised hashing approaches learn hash functions from data with supervised information to retrieve semantically similar samples. This kind of supervised information can be generated from external data other than pixels. Conventional supervised

H. Yang
School of Computer Science and Engineering, Beihang University, 37 Xueyuan Road, Haidian District, Beijing, China

X. Bai
School of Computer Science and Engineering, Beihang University, 37 Xueyuan Road, Haidian District, Beijing, China
E-mail: baixiao@buaa.edu.cn

Y. Liu
School of Computer Science and Engineering, Beihang University, 37 Xueyuan Road, Haidian District, Beijing, China

Y. Wang
School of Aeronautic Science and Engineering, Beihang University, 37 Xueyuan Road, Haidian District, Beijing, China
E-mail: wangyanyang@buaa.edu.cn

L. Bai
School of Information, Central University of Finance and Economics, 100081, Beijing, China

J. Zhou
School of Information and Communication Technology, Griffith University, Nathan, QLD 4111, Australia

W. Tang
School of Computer Science and Engineering, Beihang University, 37 Xueyuan Road, Haidian District, Beijing, China

hashing methods assume a fixed relationship between the Hamming distance and the similar (dissimilar) labels. This assumption leads to too rigid requirement in learning and makes the similar and dissimilar pairs not distinguishable. In this paper, we adopt a large margin principle and define a Hamming margin to formulate such relationship. At the same time, inspired by support vector machine which achieves strong generalization capability by maximizing the margin of its decision surface, we propose a binary hash function in the same manner. A loss function is constructed corresponding to these two kinds of margins and is minimized by a block coordinate descent method. The experiments show that our method can achieve better performance than the state-of-the-art hashing methods.

1 Introduction

Nearest neighbor (NN) search plays an important role in machine learning and information retrieval. It can be used to recognize patterns [7] or as a procedure in other algorithms. A large number of information retrieval frameworks query the most relevant instances through NN search. However, NN search is inefficient for large scale applications, and its time complexity $O(n)$ can not meets the realtime requirement. Because of this defect, hashing was introduced [12] to retrieve the approximate nearest neighbors (ANN) by mapping the data to binary embeddings. Compared with other indexing methods, hashing needs little storage and attains constant or sublinear time complexity. Early hashing methods were proposed based on certain mathematical properties to generate hash functions without any training data. These methods are called data-independent methods, in which locality sensitive hashing (LSH) [5,8] is the most representative example.

Using training data to learn hash functions can improve the hashing performance and generate compact codes [36]. This leads to the development of data-dependent hashing. Based on the training strategy, data-dependent hashing methods can be divided into unsupervised hashing [36,43,15,11,35,18,40, 10,41,24,21] and supervised hashing [19,32,27–29,23,37,42]. In addition, some methods are considered as semi-supervised hashing [34,33] because they do not require all training data to be labeled. Severa methods can adapt to both unsupervised and supervised scenario by incorporating different similarities [17, 34,25,1] or dimensionality reduction methods [9].

Unsupervised hashing method is the most basic prototype of this area. In fact, data-independent method, *i.e.* locality sensitive hashing[8] is also under the unsupervised settings. Unsupervised hashing is based on the requirement of approximate similarity search, *i.e.*, finding the nearest neighbors according to a given metric. The objective of unsupervised hashing is defined by certain similarity function, *e.g.*, the Euclidean distance. Their performance is usually evaluated based on Euclidean neighbor search. K-means hashing [10] extends the idea of product quantization to using clustering methods to generate hash functions. Spherical hashing [11] uses sphere to model hash function. In [39,

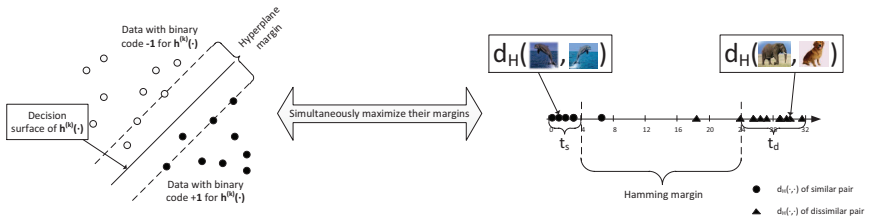


Fig. 1 Hamming margin and hyperplane margin. They are the optimization targets in our method.

38], the authors proposed a feature aggregating hashing method to achieve image copy detection on large scale data. If we can represent each image as a feature vector, *e.g.*, GIST feature [30], we can do efficient content based image retrieval by directly applying unsupervised hashing.

Recently, researchers also develop product quantization methods [13, 45] to get more accurate Euclidean neighbors than using hashing, sacrificing not much efficiency. However, the performance of using unsupervised indexing methods in image retrieval system largely relies on the image feature, which is usually cannot be optimally extracted. On the other hand, if we take use of the external data besides pixel information, we do not have this limitation any more.

Supervised hashing just follows this strategy and it aims at training hash functions which output similar binary codes for data which are labeled similar. One can use many external information to label the data, *e.g.*, the human handed annotation, the distance under other features. This kind of information is widely used in computer vision problems [44]. Therefore, supervised hashing is not only an indexing method for boosting the retrieval speed, but also can be treated as a special representation learning method [2]. Recently, by using multiple features or involving multi-view application, there are lots of hashing methods are proposed with the multimedia background [22, 14, 25, 3, 20].

This paper focuses on learning hash functions in supervised manner. Recently, some methods incorporate the specific representation into hashing framework, *e.g.*, multimodal [42] and deep convolutional network feature [37]. However, we aim at learning hash functions with any vectorized data. For supervised hashing, the supervised information is often provided as a similar set $\mathcal{S}$ and a dissimilar set $\mathcal{D}$. Each set contains a number of pairs (i, j) which indicate instances i and j are similar or dissimilar. Although the supervised information can be given in other formats (*e.g.*, class labels), they can be easily transformed to this form. If there are n instances in the training set, let Y be an $n \times r$ matrix whose i-th row is the binary code of the i-th instance, and $d_H(i, j)$ be the Hamming distance between the binary codes of i and j, then the objective is that $d_H(i, j)$ should be small for $(i, j) \in \mathcal{S}$, and $d_H(i, j)$ should be large for $(i, j) \in \mathcal{D}$. However, this objective is not definite, because it

does not define explicitly on how small (large) the Hamming distance $d_H(i,j)$ should be.

In [23], the defined objective forces the similar pairs to have the smallest Hamming distance 0 and the dissimilar pairs to have the largest Hamming distance r (the code length). In [28], Norouzi and Fleet proposed a loss function with an additional parameter ρ. It requires that the similar pairs have Hamming distance no more than ρ and the dissimilar pairs have Hamming distance no less than ρ. Nevertheless, the first definition is too strict to comply with and the second one is so slack that could make the Hamming distances of the similar pairs and dissimilar pairs not discriminative. Since each hash function maps the data to a binary code in $\{-1,+1\}$, the binary classifier is suitable for the modeling. Furthermore, because support vector machine (SVM) [6] shows high generalization capability by maximizing the margin of its decision surface, an intrinsic idea is using the similar model as the hash function. In [43], SVM is used to train hash functions, but the binary codes and hash functions are learned in two separate stages.

In this paper, we propose a novel hashing method to overcome the problems mentioned above. To well formulate the consistency of the Hamming distance and supervised affinity between each pair (i,j), we define a *Hamming margin* which equals the lower bound of $d_H(i',j')$ for all $(i',j') \in \mathcal{D}$ minus the upper bound of $d_H(i,j)$ for all $(i,j) \in \mathcal{S}$. Instead of splitting into two separate stages, our method simultaneously learns the binary codes and hash function so as to maximize the margin of the hash function. For discriminating this margin from the Hamming margin mentioned before, we refer to it as the *hyperplane margin*, and illustrate these two margins in Figure 1. Because our key idea is based on these two kinds of margin maximization, we name our method maximum margin hashing (MMH).

2 Formulation

The goal of our method is simultaneously maximizing the Hamming margin and the hyperplane margin. In this section, we formulate the loss functions to maximize these two margins respectively, and concatenate them into an integrated loss function.

2.1 Hamming Margin Loss

We first define two tolerance variables t_s and t_d. As shown in Figure 1, they control the thresholds of Hamming distance of pairs in $\mathcal{S}$ and $\mathcal{D}$. Specifically, if the code length is r, we require $d_H(i,j) \leq t_s$ for all $(i,j) \in \mathcal{S}$ and $r - d_H(i',j') \leq t_d$ for all $(i',j') \in \mathcal{D}$. To prevent the Hamming distances of pairs in S and D from intersecting, we force the Hamming margin $r - (t_s + t_d)$ to be positive, *i.e.*, $t_s + t_d < r$. Considering this constraint may be too rigid, we adopt the soft margin concept in the proposed Hamming margin. For the instance pair

(i, j), the penalization is

$$l_{ij}(t_s, t_d) = \begin{cases} [d_H(i,j) - t_s]_+ & \text{for } (i,j) \in \mathcal{S}; \\ [(r - d_H(i,j)) - t_d]_+ & \text{for } (i,j) \in \mathcal{D}. \end{cases} \tag{1}$$

where $[\alpha]_+$ is the same as $\max(0, \alpha)$. The Hamming margin loss is defined as:

$$\mathcal{L}_1 = (t_d + t_s)^2 + \eta \sum_{i<j}^{n} l_{ij}(t_s, t_d)^2 \tag{2}$$

where η is a positive number for tuning the weight of the penalization. Because the supervised information is symmetric, we only count the $i < j$ pairs in. The first term $(t_d + t_s)^2$ corresponds to Hamming margin maximization. The increase of t_m or t_d will enlarge the first term and reduce the second term, and the parameter η makes a trade off between them. What should be noted is that $d_H(i, j)$ is a function of the rows in Y, and matrix Y is also variables in our problem.

2.2 Hyperplane Margin Loss

As mentioned in the introduction section, we use the hyperplane based decision surface to model the hash function, so the k-th hash function $h^{(k)}(\cdot)$ is defined as:

$$h^{(k)}(\mathbf{x}_i) = \text{sign}(\mathbf{w}^{(k)T}\mathbf{x}_i + b^{(k)}) \tag{3}$$

where $\mathbf{w}^{(k)}$ and $b^{(k)}$ are the variables in $h^{(k)}(\cdot)$, and $\mathbf{x}_i$ is the representation vector of instance i. This definition is the same as in the kernel-based SVM. The hyperplane margin loss is analogous to the loss function used in SVM:

$$\mathcal{L}_2 = \sum_{k=1}^{r} (\frac{1}{2}\|\mathbf{w}^{(k)}\|^2 + C \sum_{i=1}^{n} [1 - Y_{ik}(\mathbf{w}^{(k)T}\mathbf{x}_i + b^{(k)})]_+) \tag{4}$$

where C is a positive parameter used to penalize the violation caused by using the soft margin, Y_{ik} is the element in i-th row, k-th column of Y, and $\mathcal{H} = \{h^{(k)}(\cdot)\}_{k=1}^{r}$ is the set of r hash functions. The difference between our objective in equation (4) and the objective in SVM is that for $i = 1, 2, ..., n$, Y_{ik} are also variables that need to be optimized, so our objective is a more complex problem.

Finally, our integrated objective is minimizing the Hamming margin loss in equation (2) and the hyperplane margin loss in equation (4) simultaneously. We combines them as a weighted sum:

$$\begin{aligned} \min_{\mathcal{H}, Y, t_s, t_d} \quad & \lambda\mathcal{L}_1 + \mathcal{L}_2 \\ \text{subject to} \quad & Y \in \{-1, +1\}^{n \times r}, \\ & t_s, t_d \in \{0, 1, ..., r\}, t_s + t_d < r \end{aligned} \tag{5}$$

where positive parameter λ provides the trade off between these two margins.

3 Optimization

In Section 2, we have presented the objective (5). It is not differentiable because of the $[\cdot]_+$ operator. Moreover, it has a large number of variables including real-valued numbers, nonnegative integers, and binary numbers. This leads to a complex and discontinuous feasible set. It is difficult to find an efficient method that can solve it directly. In this section, we propose to use the block coordinate descent (BCD) approach and enable some relaxation to get a good solution. The principle of BCD is separating the variables into several groups, and optimizing the problem with respect to each group of variables iteratively.

3.1 Optimizing with Fixed Tolerance Variables

Firstly, we assume the tolerance variables t_s and t_d are constant, *i.e.*, $t_s = \hat{t}_s, t_d = \hat{t}_d$. Let $\mathbf{y}^{(k)}$ corresponds to the k-th column of Y. Since the structure of each hash function and hash bit is the same, we group them to use the BCD. After removing the constant terms, the subproblem is:

$$\begin{aligned} &\min_{\mathbf{w}^{(k)}, b^{(k)}, \mathbf{y}^{(k)}} \lambda \cdot \eta \sum_{i<j}^{n} l_{ij}(\hat{t}_s, \hat{t}_d)^2 + \\ &\frac{1}{2}\|\mathbf{w}^{(k)}\|^2 + C \sum_{i=1}^{n} [1 - \mathbf{y}_i^{(k)}(\mathbf{w}^{(k)T}\mathbf{x}_i + b^{(k)})]_+ \end{aligned} \tag{6}$$

Then we can iteratively optimize the subproblems with $k = 1, 2, ..., r$ until Y converges. This is also a difficult problem, so we simplify it by using BCD again.

Fix $\mathbf{y}^{(k)}$ and update $\mathbf{w}^{(k)}, b^{(k)}$, *i.e.*, $\mathbf{y}_i^{(k)} = \hat{\mathbf{y}}_i^{(k)}$. After fixing $\mathbf{y}^{(k)}$, solving the optimal $\mathbf{w}^{(k)}, b^{(k)}$ is just the same problem in SVM:

$$\operatorname*{argmin}_{\mathbf{w}^{(k)}, b^{(k)}} \frac{1}{2}\|\mathbf{w}^{(k)}\|^2 + C \sum_{i=1}^{n} [1 - \hat{\mathbf{y}}_i^{(k)}(\mathbf{w}^{(k)T}\mathbf{x}_i + b^{(k)})]_+ \tag{7}$$

There are many SVM solvers that can be used to solve this optimization problem. Here we use its subgradient and L-BFGS method for efficiency.

Fix $\mathbf{w}^{(k)}, b^{(k)}$ and update $\mathbf{y}^{(k)}$, *i.e.*, $\mathbf{w}^{(k)} = \hat{\mathbf{w}}^{(k)}, b^{(k)} = \hat{b}^{(k)}$. This situation is more complex. We first simplify the terms with the following propositions.

Proposition 1 *Let $l_{ij}^{(\bar{k})}(\hat{t}_s, \hat{t}_d) = l(d_H^{(\bar{k})}(i,j); \hat{t}_s, \hat{t}_d)$ denote the penalization in equation (1) between i and j excluding the k-th bit, where $d_H^{(\bar{k})}(i,j)$ is the Hamming distance excluding the k-th bit, then we have:*

$$\sum_{i<j}^{n} l_{ij}(\hat{t}_s, \hat{t}_d)^2 = \sum_{i<j}^{n} (l_{ij}^{(\bar{k})}(\hat{t}_s, \hat{t}_d)^2 + \frac{1}{2} W_{ij}(1 - \mathbf{y}_i^{(k)}\mathbf{y}_j^{(k)})) \tag{8}$$

where W is an $n \times n$ matrix and $W_{ij} = l(d_H^{(\bar{k})}(i,j)+1; \hat{t}_s, \hat{t}_d)^2 - l(d_H^{(\bar{k})}(i,j); \hat{t}_s, \hat{t}_d)^2$.

Proof It is obvious that $l(d_H(i,j);\hat{t}_s,\hat{t}_d)^2 =$

$$\begin{cases} l(d_H^{(\bar{k})}(i,j);\hat{t}_s,\hat{t}_d)^2 + W_{ij}\cdot 0, \ \mathbf{y}_i^{(k)} = \mathbf{y}_j^{(k)}; \\ l(d_H^{(\bar{k})}(i,j);\hat{t}_s,\hat{t}_d)^2 + W_{ij}\cdot 1, \ \mathbf{y}_i^{(k)} \neq \mathbf{y}_j^{(k)}. \end{cases}$$

Merging these two situations into one expression, we get equation (8).

Proposition 2 *Let* $\gamma_i = \hat{\mathbf{w}}^{(k)T}\mathbf{x}_i + \hat{b}^{(k)}$, *then* $\sum_{i=1}^{n}[1-\mathbf{y}_i^{(k)}(\hat{\mathbf{w}}^{(k)T}\mathbf{x}_i+\hat{b}^{(k)})]_+ =$

$$\begin{aligned} -\sum_{|\gamma_i|<1}\gamma_i\mathbf{y}_i^{(k)} - \frac{1}{2}\sum_{|\gamma_i|\geq 1}(sign(\gamma_i)+\gamma_i)\mathbf{y}_i^{(k)} \\ + \sum_{|\gamma_i|<1} 1 + \frac{1}{2}\sum_{|\gamma_i|\geq 1}(1+|\gamma_i|) \end{aligned} \tag{9}$$

Proof It is easy to get the piecewise formulation that $[1-\mathbf{y}_i^{(k)}(\hat{\mathbf{w}}^{(k)T}\mathbf{x}_i + \hat{b}^{(k)})]_+ =$

$$\begin{cases} 1+|\gamma_i| & \text{for } |\gamma_i|<1, \text{ and } \mathbf{y}_i^{(k)}\text{sign}(\gamma_i) = -1; \\ 1-|\gamma_i| & \text{for } |\gamma_i|<1, \text{ and } \mathbf{y}_i^{(k)}\text{sign}(\gamma_i) = 1; \\ 1+|\gamma_i| & \text{for } |\gamma_i|\geq 1, \text{ and } \mathbf{y}_i^{(k)}\text{sign}(\gamma_i) = -1; \\ 0 & \text{for } |\gamma_i|\geq 1, \text{ and } \mathbf{y}_i^{(k)}\text{sign}(\gamma_i) = 1. \end{cases}$$

Merge it as:

$$\begin{cases} 1-\text{sign}(\gamma_i)\mathbf{y}_i^{(k)}|\gamma_i| & \text{for } |\gamma_i|<1; \\ \frac{1}{2}(1+|\gamma_i|)(1-\text{sign}(\gamma_i)\mathbf{y}_i^{(k)}) & \text{for } |\gamma_i|\geq 1. \end{cases}$$

Using the above equation for summing, we get the equation (9).

With the Propositions 1, 2 and the fact that $l_{ij}^{(\bar{k})}(\hat{t}_s,\hat{t}_d)$ and γ_i are constants here, the subproblem defined in equation (6) with fixed $\mathbf{w}^{(k)}, b^{(k)}$ becomes:

$$\begin{aligned} &\min_{\mathbf{y}^{(k)}}\{\lambda\cdot\eta\sum_{i<j}^{n}\frac{1}{2}W_{ij}(1-\mathbf{y}_i^{(k)}\mathbf{y}_j^{(k)}) + C(-\sum_{|\gamma_i|<1}\gamma_i\mathbf{y}_i^{(k)} \\ &-\frac{1}{2}\sum_{|\gamma_i|\geq 1}(\text{sign}(\gamma_i)+\gamma_i)\mathbf{y}_i^{(k)})\} \end{aligned}$$

Furthermore, we can represent this problem as an unconstrained binary quadratic programming (BQP):

$$\begin{aligned} \min_{\mathbf{y}^{(k)}} \quad & \frac{1}{2}\mathbf{y}^{(k)T}L\mathbf{y}^{(k)} + \mathbf{a}^T\mathbf{y}^{(k)} \\ \text{subject to} \quad & \mathbf{y}^{(k)} \in \{-1,+1\}^n \end{aligned} \tag{10}$$

with the $n \times n$ matrix L and n dimensional vector $\mathbf{a}$:

$$\begin{aligned} L &= \frac{1}{2}\lambda \cdot \eta(\mathrm{Diag}(W\mathbf{e}) - W), \\ \mathbf{a}_i &= \begin{cases} -C\gamma_i & \text{for } |\gamma_i| < 1; \\ -C(\mathrm{sign}(\gamma_i) + \gamma_i)/2 & \text{for } |\gamma_i| \geq 1. \end{cases} \end{aligned} \tag{11}$$

where $\mathrm{Diag}(W\mathbf{e})$ is a diagonal matrix with the i-th diagonal element being the sum of the i-th row in W.

Although equation (10) has a very compact form, it is an NP-hard problem [31]. Some relaxation solutions have been proposed for it [31]. To optimize our integrated objective, we need to solve this kind of BQP many times, so the efficiency of the BQP solver is crucial. In this paper, we use a simple approximation:

$$\min_{\mathbf{z}} \quad \frac{1}{2}\tanh(\mathbf{z}^T)L\tanh(\mathbf{z}) + \mathbf{a}^T\tanh(\mathbf{z}) \tag{12}$$

where $\tanh(\cdot)$ is used as a smooth approximation of $\mathrm{sign}(\cdot)$.

Equation (12) is an unconstrained optimization problem, so we use its gradient and the L-BFGS method to optimize it. The binary variables can be obtained by $\mathrm{sign}(\mathbf{z}^*)$, where $\mathbf{z}^*$ is the minimum point. Our solution iteratively updates $\mathbf{w}^{(k)}, b^{(k)}$ and $\{\mathbf{y}_i^{(k)}\}_{i=1}^n$ until they converge, then goes on to the optimization with the $(k+1)$-th hash function and bit.

3.2 Updating the Tolerance Variables

Following the above procedures, we can get $\mathcal{H}$ and Y in equation (5) with fixed t_s and t_d. Here we use the BCD principle again to update these two variables.

If the binary code matrix Y is given, the Hamming distances are constant, *i.e.*, $d_H(i,j) = \hat{d}_H(i,j)$. It should be noted that the variables t_s and t_d only appear in $\mathcal{L}_1$, so the subproblem becomes nonlinear integer programming:

$$\begin{aligned} \min_{t_s,t_d} \quad & (t_d + t_s)^2 + \eta\Big(\sum_{(i,j)\in\mathcal{S}} [\hat{d}_H(i,j) - t_s]_+^2 + \\ & \sum_{(i,j)\in\mathcal{D}} [r - \hat{d}_H(i,j) - t_d]_+^2\Big) \\ \text{subject to} \quad & t_s, t_d \in \{0,1,...,r\}, t_s + t_d < r \end{aligned} \tag{13}$$

The objective function is a nonnegative weighted sum of several convex functions, so it is also a convex function. However, its feasible set is a triangular lattice which is discrete. Here we propose a steepest direction search method which is analogous to gradient descent, but is applicable to discrete optimization.

Let f denote the objective function in (13), the steepest direction search method consists of following steps:

1. Select four points $(t_s \pm 1, t_d)$ and $(t_s, t_d \pm 1)$, and choose the feasible points (maintain the constraint in equation (13)) in them as candidates;
2. Compute the value of f on the candidates, and choose the minimal point (t_s^*, t_d^*);
3. If $f(t_s^*, t_d^*) < f(t_s, t_d)$, update (t_s, t_d) with (t_s^*, t_d^*) and go to step 1. Otherwise stop and return (t_s, t_d).

With the above procedure, we can get the point (t_s, t_d) which has the minimal f in its adjacencies $(t_s \pm 1, t_d)$ and $(t_s, t_d \pm 1)$. Furthermore, because of the convexity of function f, we have $f(t_s, t_d) \leq f(t_s \pm t, t_d)$ and $f(t_s, t_d) \leq f(t_s, t_d \pm t)$, where t is an arbitrary integer. Although this is not a global optimal solution, it is usually good enough for our problem. After updating t_s, t_d, we can optimize $\mathcal{H}, Y$ according to Section 3.1. Finally, the whole procedure for solving our objective in equation (5) is shown in Algorithm 1.

Algorithm 1: Maximum Margin Hashing

Data: Representation vectors $\{\mathbf{x}_i\}_{i=1}^n$, similar set $\mathcal{S}$, dissimilar set $\mathcal{D}$, code length r and parameters λ, η, C.
Result: Hash function set $\mathcal{H}$.
Initialize t_s, t_d and binary code matrix $Y \in \{-1, +1\}^{n \times r}$.
repeat `// Iteration 1`
  $k \leftarrow 1$.
  repeat `// Iteration 2`
    Construct matrix $L \in \mathbb{R}^{n \times n}$ as equation (11);
    repeat `// Iteration 3`
      Construct vector $\mathbf{a} \in \mathbb{R}^n$ as equation (11) or initiate $\mathbf{a}$ as $\mathbf{0}$ in the first round;
      Solve the local minima $\mathbf{z}^*$ of the unconstrained problem (12);
      $\mathbf{y}_i^{(k)} \leftarrow \text{sign}(\mathbf{z}_i^*)$, for $i = 1, 2, 3, ..., n$;
      Solve the optimal $\mathbf{w}^{(k)}, b^{(k)}$ of the SVM problem (7) with data $\mathbf{x}_i$ and labels $\mathbf{y}_i^{(k)}$;
    until $\{\mathbf{y}_i^{(k)}\}_{i=1}^n$, $\mathbf{w}^{(k)}, b^{(k)}$ *are converged*;
    $Y_{ik} \leftarrow \mathbf{y}_i^{(k)}, i = 1, 2, 3, ..., n$;
    $k \leftarrow k + 1$;
    if $k > r$ **then**
      $k \leftarrow 1$
    end
  until Y *is converged*;
  Update t_s, t_d according to the steepest direction search in Section 3.2.
until t_s, t_d *are converged*;
$\mathcal{H} \leftarrow \{h^{(k)}(\cdot)\}_{k=1}^r$. `//` $h^{(k)}(\mathbf{x}) = \texttt{sign}(\mathbf{w}^{(k)T}\mathbf{x} + b^{(k)})$

4 Implementation Details

Initialization. Some variables in Algorithm 1 need to be initialized. At the beginning of the whole process, we initialize $t_s = 0, t_d = 0$ so as to start with the largest Hamming margin. The matrix Y is initialized with random

projections of $\{\mathbf{x}_i\}_{i=1}^n$. In the first round, since $\mathbf{w}^{(k)}, b^{(k)}$ are not given, we can initiate $\mathbf{a} = \mathbf{0}$. To proceed the L-BFGS method for equation (12), we also need an initial point $\mathbf{z}$. Here we set it as $\sqrt{n} \cdot \mathbf{u}$ where $\mathbf{u}$ is the eigenvector of L corresponding to the least eigenvalue and $\mathbf{a}^T\mathbf{u} < 0$. At the end of iteration 1, we use the previous t_s and t_d to start the steepest direction search.

Convergency. In Algorithm 1, all stop criteria are the variables to be converged. For the BCD method, the value of objective function will decrease after each iteration if the global minima of the subproblem is attained. In our method, we usually can not get the global minima, but we can guarantee that the value of objective function does not increase after each iteration. In practice, if the solved variables for the subproblem will increase the overall loss, we do not update the corresponding variables. Therefore, the variables will be converged when the loss is small enough.

Parameters. The main parameters in our method are the tradeoff parameters λ, η and C. λ controls the balance between $\mathcal{L}_1$ and $\mathcal{L}_2$, and C is similar to the soft margin weight in SVM. As in many algorithms, these parameters need to be tuned on different datasets. In our experiments, we set $\eta = n(n-1)/2$, and tuned λ and C.

5 Experiments

5.1 Experiments Set up

Datasets. We evaluated the performance of the proposed method on two popular image datasets, MNIST[1], CIFAR-10 [16] and one multi-modal dataset Wiki[2]. They are widely used in hashing evaluations [17,34,27,28,33,23,9].

MNIST dataset consists of 70,000 28×28 gray scale images of handwritten digits. This dataset has 10 classes corresponding to the 0∼9 digits, with all images being labeled. The digit in each image is well aligned, so the pixel values can be treated as a 784-dimension feature vector. Because a large portion of pixels are clean background pixels in each image, each feature vector is sparse.

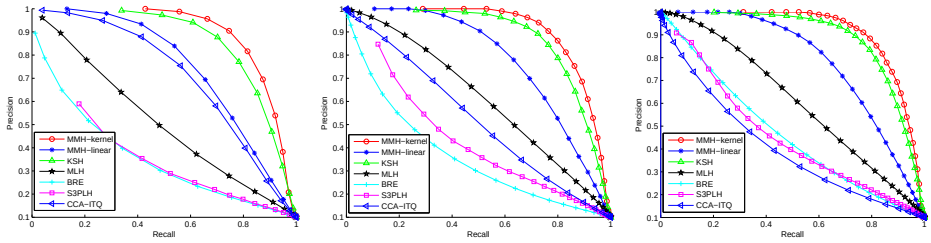
CIFAR-10 dataset consists of 60,000 32×32 color images in 10 classes including airplane, automobile, bird, cat, deer, dog, frog, horse, ship, and truck. Each class has 6,000 images. For CIFAR-10, the images are too small to extract adequate local features such as SIFT [26]. Since they have the same size, we use a 512-dimensional GIST descriptor [30] to represent each image.

Wiki dataset contains images and texts from Wikipedia. The size of whole dataset is 2,886, and every sample is a image-text pair. In our experiment, we use the 128-dimensional SIFT based bag-of-visual word representation for the image, and 10-dimensional topic vector for the text content.

For the two image datasets, the image label is actually the supervised information which can be seen as the external data for pixel data of images. For dataset Wiki, we compute the Euclidean distance between the texts' features

[1] http://yann.lecun.com/exdb/mnist/

[2] http://www.svcl.ucsd.edu/projects/crossmodal/



(a) 16 bits (b) 32 bits (c) 48 bits

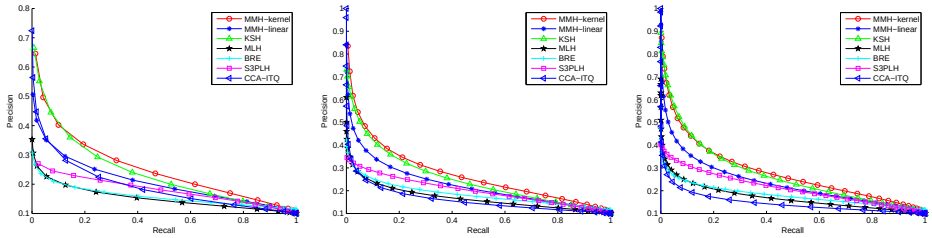
Fig. 2 Precision-recall curves on MNIST.

and set an threshold τ to decide whether two images are similar($< \tau$) or dissimilar($\geq \tau$).

Evaluation Protocols and Baseline Methods. In the experiments, we assess the proposed maximum margin hashing (MMH) method. Each image has a single class label in MNIST and CIFAR-10. Images in the same class are considered as the semantic neighbors to each other. For fair comparison, we compare our methods with some state-of-the-art supervised hashing methods, which include kernel-based supervised hashing (KSH) [23], minimal loss hashing (MLH) [28], binary reconstructive embedding (BRE) [17], semi-supervised sequential projection learning hashing (S3PLH) [34], and iterative quantization based on CCA (CCA-ITQ) [9]. In the end of the training procedure, we can learn the kernelized hash function by LIBSVM [4] with labels in Y. We denote the linear version and kernelized version as MMH-linear and MMH-kernel, respectively. For the methods which use kernel-based hash functions, we adopt the same Gaussian RBF kernel $\exp(-\|\mathbf{x}_i - \mathbf{x}_j\|^2/\sigma)$, where σ is set as the mean of squared Euclidean distances of the training set. For MNIST and CIFAR-10, we randomly choose 1,000 samples as the query images, and use the rest of the dataset as the target of the search, and for Wiki, the number of query is 100. For MMH,KSH,MLH, and BRE, the scale of the training set can not be too large because they are based on pairwise supervised information, we randomly choose 1,000 instances for training. S3PLH is a semi-supervised method, we choose 1,000 labeled images and take the rest images in training set as unlabeled for training. CCA-ITQ can be trained on large scale dataset since its training time is mainly dependent on the dimensionality of the data, so we randomly choose 50,000 instances for training. All the other parameters are set according to the original papers reporting these methods.

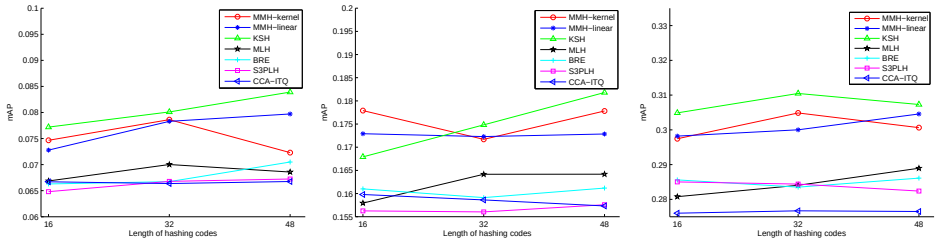
5.2 Evaluation of Retireval Performance

Evaluation with the Overall Measure. To demonstrate the overall performance, the precision-recall curves are shown in Figures 2 and 3. For the given queries, we increase the Hamming radius from 0 to r, and compute the precision and recall values of the retrieved instances, then the precision-recall



(a) 16 bits (b) 32 bits (c) 48 bits

Fig. 3 Precision-recall curves on CIFAR-10.



(a) $\tau = 0.2$ (b) $\tau = 0.3$ (c) $\tau = 0.4$

Fig. 4 Mean Average Precision with different parameter τ on Wiki.

curve is plotted according to the $r + 1$ points. It can be seen that our method MMH-kernel achieves the best results on both datasets, while the superiority on MNIST is more significant. Using the linear hash functions, our method MMH-linear has a lower performance compared with MMH-kernel and another kernel-based method KSH, but outperforms the others. Almost all methods get better results when using longer hash codes, except for CCA-ITQ. The reason is that CCA-ITQ is based on the Canonical Correlation Analysis which is for generating low dimensional output, so the useful information will be diluted if too long hash codes are used. For all the methods, the performances on MNIST are better than on CIFAR-10. We think this is due to the differences on low level representations and the difficulty of the retrieval tasks. For dataset Wiki, we show the mean average precision (mAP) in Figure 4, which is the area under the precision-recall curve, with different value of τ.

Evaluation for the Practical Usage. In practice, there are two major applications for the hash codes, *i.e.* Hamming ranking and hash lookup. Hamming ranking compares the hash code of the query with all samples in the database, which leads to linear complexity but can be efficient thanks to the efficiency of comparison between binary codes. Hash lookup retrieves samples that fall in the buckets within a given Hamming radius. In Table 1, we show the mean precisions (MP) for the top 500 Hamming neighbors (for Hamming ranking) and samples whose Hamming distances to the query is less than 2 (for hash lookup). In most situations, our method MMH-kernel has the highest

Table 1 Mean precisions and on MNIST and CIFAR-10.

Methods\#bits	**MP@top500**			**MP@$d_H < 2$**		
	16	32	48	16	32	48
MMH-kernel	**0.885**	**0.894**	**0.901**	**0.883**	**0.874**	**0.839**
MMH-linear	0.794	0.826	0.835	0.801	0.788	0.647
KSH	0.862	0.892	0.894	0.866	0.851	0.752
MLH	0.660	0.759	0.782	0.702	0.696	0.424
BRE	0.575	0.622	0.721	0.639	0.637	0.378
S3PLH	0.446	0.554	0.599	0.397	0.557	0.648
CCA-ITQ	0.796	0.742	0.671	0.814	0.155	0.004

(a) MNIST

Methods\#bits	**MP@top500**			**MP@$d_H < 2$**		
	16	32	48	16	32	48
MMH-kernel	**0.355**	**0.386**	**0.406**	0.354	**0.362**	0.255
MMH-linear	0.301	0.323	0.344	0.306	0.270	0.126
KSH	0.351	0.371	0.395	0.352	0.223	0.098
MLH	0.220	0.246	0.265	0.245	0.091	0.011
BRE	0.198	0.231	0.232	0.213	0.176	0.069
S3PLH	0.253	0.288	0.310	0.240	0.296	**0.298**
CCA-ITQ	0.323	0.265	0.245	**0.371**	0.012	0.001

(b) CIFAR-10

precision. When the hash code length r is too large, the performance of hash lookup may be worse with longer codes because the hash table becomes too sparse in this situation.

Table 2 Training times (seconds) on MNIST and CIFAR-10.

Methods\#bits	Training time on MNIST			Training time on CIFAR-10		
	16	32	48	16	32	48
MMH-kernel	17.62	41.81	58.62	21.61	36.26	41.66
MMH-linear	15.44	34.21	47.08	20.15	35.33	38.41
KSH	32.07	67.61	105.6	32.95	64.98	104.84
MLH	438.1	2279	3545	465.7	2184	3476
BRE	34.29	65.96	158.1	48.58	114.4	270.4
S3PLH	18.24	35.50	53.36	9.389	18.17	28.47
CCA-ITQ	2.507	3.392	4.326	1.719	2.628	3.554

5.3 Time Efficiency Comparison

Since the specific number of iterations is not determined, the theoretical training complexity of our method is not definite. In Algorithm 1, we can see there are three kinds of iterations. Usually, iteration 1 and 3 only repeat less than ten times. Iteration 2 usually repeats less than $2r$ times. In our implementation, we limit the maximal number of iteration of solving problem 12 and 7. The elapsed time for training are shown in Tables 2. All the experiments are conducted on a PC with Core-i7 3.4GHz CPU and 16GB RAM. We can see that our methods MMH-kernel and MMH-linear consume less time than most methods, and only longer than CCA-ITQ and S3PLH whose retrieval performances are much lower.

In the indexing procedure, for the methods using linear hash function, *i.e.*, MMH-linear, MLH, S3PLH, and CCA-ITQ, the time complexity of each hash function is $O(d)$, where d is the dimensionality of data. Methods MMH-kernel, KSH, and BRE use kernelized hash function and the time complexity of Gaussian kernel is $O(d)$, so the time complexity of their hash function is $O(\mu d + \mu)$, where μ is the number of the support vectors.

6 Conclusion

In this paper, we present a supervised hashing based on two kinds of margins, *i.e.* the Hamming margin and the hyperplane margin. Our Hamming margin loss is a more general definition of the requirement on the relationship between Hamming distance and the supervised labels, whereas many other methods only use a fixed Hamming margin. At the same time, we require the decision surface of hash function having a large margin to enhance the generalization capability as SVM. Experiments have shown the superiority of our approach.

7 Acknowledge

This work was supported by NSFC projects (No. 61370123 and 61503422), and the Australian Research Councils DECRA Projects funding scheme (project ID 522 DE120102948).

References

1. Bai, X., Yang, H., Zhou, J., Ren, P., Cheng, J.: Data-dependent hashing based on p-stable distribution. Image Processing, IEEE Transactions on **23**(12), 5033–5046 (2014)
2. Bengio, Y., Courville, A., Vincent, P.: Representation learning: A review and new perspectives. IEEE Transactions on Pattern Analysis and Machine Intelligence **35**(8), 1798–1828 (2013)
3. Bronstein, M.M., Bronstein, A.M., Michel, F., Paragios, N.: Data fusion through cross-modality metric learning using similarity-sensitive hashing. In: Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on, pp. 3594–3601. IEEE (2010)
4. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines. ACM Transactions on Intelligent Systems and Technology **2**, 27:1–27:27 (2011)
5. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing, pp. 380–388. ACM (2002)
6. Cortes, C., Vapnik, V.: Support-vector networks. Machine learning **20**(3), 273–297 (1995)
7. Cover, T., Hart, P.: Nearest neighbor pattern classification. IEEE Transactions on Information Theory **13**(1), 21–27 (1967)
8. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proceedings of the Annual Symposium on Computational Geometry, pp. 253–262 (2004)
9. Gong, Y., Lazebnik, S., Gordo, A., Perronnin, F.: Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. IEEE Transactions on Pattern Analysis and Machine Intelligence **35**(12), 2916–2929 (2013)

10. He, K., Wen, F., Sun, J.: K-means hashing: an affinity-preserving quantization method for learning binary compact codes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2938–2945 (2013)
11. Heo, J.P., Lee, Y., He, J., Chang, S.F., Yoon, S.E.: Spherical hashing. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2957–2964 (2012)
12. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the Annual ACM symposium on Theory of computing, pp. 604–613 (1998)
13. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. Pattern Analysis and Machine Intelligence, IEEE Transactions on **33**(1), 117–128 (2011)
14. Jiang, Y.G., Wang, J., Xue, X., Chang, S.F.: Query-adaptive image search with hash codes. Multimedia, IEEE Transactions on **15**(2), 442–453 (2013)
15. Kong, W., Li, W.J.: Isotropic hashing. In: Proceedings of the Neural Information Processing Systems Conference, pp. 1655–1663 (2012)
16. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Master's thesis, Department of Computer Science, University of Toronto (2009)
17. Kulis, B., Darrell, T.: Learning to hash with binary reconstructive embeddings. Proceedings of the Neural Information Processing Systems Conference **22**, 1042–1050 (2009)
18. Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing. IEEE Transactions on Pattern Analysis and Machine Intelligence **34**(6), 1092–1104 (2012)
19. Kulis, B., Jain, P., Grauman, K.: Fast similarity search for learned metrics. IEEE Transactions on Pattern Analysis and Machine Intelligence **31**(12), 2143–2157 (2009)
20. Kumar, S., Udupa, R.: Learning hash functions for cross-view similarity search. In: IJCAI Proceedings-International Joint Conference on Artificial Intelligence, vol. 22, p. 1360 (2011)
21. Leng, C., Wu, J., Cheng, J., Bai, X., Lu, H.: Online sketching hashing. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2503–2511 (2015)
22. Li, P., Wang, M., Cheng, J., Xu, C., Lu, H.: Spectral hashing with semantically consistent graph for image indexing. IEEE Transactions on Multimedia **15**(1), 141–152 (2013)
23. Liu, W., Wang, J., Ji, R., Jiang, Y., Chang, S.: Supervised hashing with kernels. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 2074–2081 (2012)
24. Liu, X., Du, B., Deng, C., Liu, M., Lang, B.: Structure sensitive hashing with adaptive product quantization. IEEE Transactions on Cybernetics **PP**(0), 1–12 (2015)
25. Liu, X., He, J., Lang, B.: Multiple feature kernel hashing for large-scale visual search. Pattern Recognition **47**(2), 748–757 (2014)
26. Lowe, D.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision **60**(2), 91–110 (2004)
27. Mu, Y., Shen, J., Yan, S.: Weakly-supervised hashing in kernel space. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 3344–3351 (2010)
28. Norouzi, M., Fleet, D.J.: Minimal loss hashing for compact binary codes. In: Proceedings of the International Conference on Machine Learning, pp. 353–360 (2011)
29. Norouzi, M., Fleet, D.J., Salakhutdinov, R.: Hamming distance metric learning. In: Proceedings of the Neural Information Processing Systems Conference, pp. 1070–1078 (2012)
30. Oliva, A., Torralba, A.: Modeling the shape of the scene: A holistic representation of the spatial envelope. International Journal of Computer Vision **42**(3), 145–175 (2001)
31. Olsson, C., Eriksson, A.P., Kahl, F.: Solving large scale binary quadratic problems: Spectral methods vs. semidefinite programming. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8. IEEE (2007)
32. Salakhutdinov, R., Hinton, G.: Semantic hashing. International Journal of Approximate Reasoning **50**(7), 969–978 (2009)
33. Wang, J., Kumar, S., Chang, S.: Semi-supervised hashing for large scale search. IEEE Transactions on Pattern Analysis and Machine Intelligence **34**(12), 2393–2406 (2012)

34. Wang, J., Kumar, S., Chang, S.F.: Sequential projection learning for hashing with compact codes. In: Proceedings of the International Conference on Machine Learning, pp. 1127–1134 (2010)
35. Weiss, Y., Fergus, R., Torralba, A.: Multidimensional spectral hashing. In: Proceedings of the European Conference on Computer Vision, pp. 340–353 (2012)
36. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: Proceedings of the Neural Information Processing Systems Conference, pp. 1753–1760 (2008)
37. Xia, R., Pan, Y., Lai, H., Liu, C., Yan, S.: Supervised hashing for image retrieval via image representation learning. In: Twenty-Eighth AAAI Conference on Artificial Intelligence (2014)
38. Yan, L., Ling, H., Ye, D., Wang, C., Ye, Z., Chen, H.: Feature fusion based hashing for large scale image copy detection. International Journal of Computational Intelligence Systems **8**(4), 725–734 (2015)
39. Yan, L., Zou, F., Guo, R., Gao, L., Zhou, K., Wang, C.: Feature aggregating hashing for image copy detection. World Wide Web pp. 1–13 (2015)
40. Yang, H., Bai, X., Liu, C., Zhou, J.: Label propagation hashing based on p-stable distribution and coordinate descent. In: Proceedings of the International Conference on Image Processing (2013)
41. Yang, H., Bai, X., Zhou, J., Ren, P., Zhang, Z., Cheng, J.: Adaptive object retrieval with kernel reconstructive hashing. In: Computer Vision and Pattern Recognition (CVPR), 2014 IEEE Conference on, pp. 1955–1962. IEEE (2014)
42. Zhang, D., Li, W.J.: Large-scale supervised multimodal hashing with semantic correlation maximization. In: Proceedings of the AAAI Conference on Artificial Intelligence (2014)
43. Zhang, D., Wang, J., Cai, D., Lu, J.: Self-taught hashing for fast similarity search. In: Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval, pp. 18–25 (2010)
44. Zhang, H., Bai, X., Zhou, J., Cheng, J., Zhao, H.: Object detection via structural feature selection and shape model. Image Processing, IEEE Transactions on **22**(12), 4984–4995 (2013)
45. Zhang, T., Du, C., Wang, J.: Composite quantization for approximate nearest neighbor search. In: Proceedings of the 31st International Conference on Machine Learning (ICML-14), pp. 838–846 (2014)