Class-wise Supervised Hashing with Label Embedding and Active Bits

Long-Kai Huang and Sinno Jialin Pan
Nanyang Technological University, Singapore
lhuang018@e.ntu.edu.sg, sinnopan@ntu.edu.sg

Abstract

Learning to hash has become a crucial technique for big data analytics. Among existing methods, supervised learning approaches play an important role as they can produce compact codes and enable semantic search. However, the size of an instance-pairwise similarity matrix used in most supervised hashing methods is quadratic to the size of labeled training data, which is very expensive in terms of space, especially for a large-scale learning problem. This limitation hinders the full utilization of labeled instances for learning a more precise hashing model. To overcome this limitation, we propose a class-wise supervised hashing method that trains a model based on a class-pairwise similarity matrix, whose size is much smaller than the instance-pairwise similarity matrix in many applications. In addition, besides a set of hash functions, our proposed method learns a set of class-wise code-prototypes with active bits for different classes. These class-wise code-prototypes can help to learn more precise compact codes for semantic information retrieval. Experimental results verify the superior effectiveness of our proposed method over other baseline hashing methods.

1 Introduction

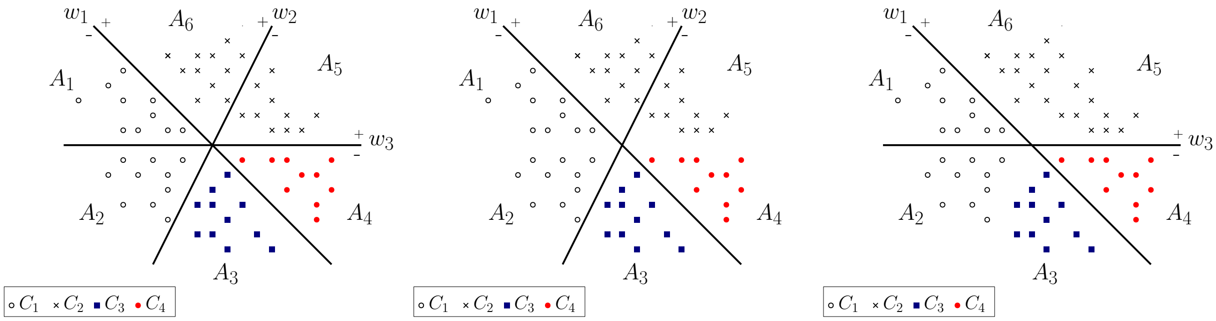
The hashing technique has received more and more attention in large-scale data analytics applications because it can save storage space by mapping each raw data into a binary code and enable fast search. Data-independent hashing methods, such as Locality Sensitive Hashing (LSH) [Gionis *et al.*, 1999; Charikar, 2002], require a long-length binary code to guarantee precision in searching using approximate nearest neighbor (ANN). In order to produce a more compact code, data-dependent hashing methods, known as learning to hash methods [Wang *et al.*, 2016; 2014] have been proposed. Compared to unsupervised hashing methods [Weiss *et al.*, 2008; Liu *et al.*, 2014; 2011; Gong and Lazebnik, 2011], which do not take label information into consideration when learning hash functions, supervised and semi-supervised approaches [Shen *et al.*, 2015; Wang *et al.*, 2010a; 2010b; Liu *et al.*, 2012a; Pan *et al.*, 2015; Lin *et al.*, 2014] are more promising as they can encode side information for semantic search.

For training hashing functions, most existing supervised and semi-supervised methods [Wang *et al.*, 2010a; 2010b; Liu *et al.*, 2012a; Pan *et al.*, 2015; Lin *et al.*, 2014] rely on an instance-pairwise similarity matrix of labeled training data. The cost in terms of space for generating this matrix is quadratic to the size of the labeled training data. In a large-scale learning problem, the number of labeled training instances can be millions or even billions. In this case, it is impossible to generate a full instance-pairwise similarity matrix for training. As a compromise, sampling techniques are usually adopted to select only a small set of labeled instances to compute the similarity matrix while the remaining labeled instance are discarded [Liu *et al.*, 2012a] or used as "unlabeled data" to construct a regularization term for learning [Wang *et al.*, 2010a; 2010b; Pan *et al.*, 2015; Lin *et al.*, 2014]. However, this may result in information loss for learning semantic hashing functions.

To address the issue mentioned above, in this paper, we propose a class-wise supervised hashing method, which trains a model based on a class-pairwise similarity matrix rather than an instance-pairwise similarity matrix. In many applications, the number of classes is much smaller than that of training instances. Therefore, the storage space for the similarity matrix is dramatically reduced from $O(N^2)$ to $O(L^2)$, where N is total number of labeled instance, and L is total number of classes, which is supposed to be much smaller than N. With the class-pairwise similarity matrix, besides a set of hash functions to map original instances to binary codes[1], we also aim to learn a class-wise code-prototype for each class with "active" bits.

Our idea is motivated by the example shown in Figure 1, where we are given a set of training instances of four classes: C_1, C_2, C_3 and C_4. The goal is to learn three linear hash functions in terms of w_1, w_2 and w_3 to map an instance to a code of three binary bits. Ideally, instances belonging to the same class are expected to have very similar hash codes, while instances belonging to different classes are expected to have very different codes. However, when the number of classes and data size of each class are both large, one may not be able to find a perfect set of hash functions over all the

[1] In this paper, we use $+1$ and -1 to construct binary codes.



(a) Using all the three hashing functions (b) Make w_3 inactive for class C_1 (c) Make w_2 inactive for class C_2

Figure 1: An motivating example for active hashing functions

classes through optimization. Suppose the optimal solutions for w_1, w_2 and w_3 are shown in Figure 1(a), where the instances of class C_1 are divided by w_3 into two areas: A_1 and A_2, while the instances of class C_2 are divided by w_2 into another two areas: A_5 and A_6. Suppose given a query of class C_1, which falls in area A_2, only instances in area A_2 will be turned within Hamming radius 0. If we relax the retrieval condition to be within Hamming radius 1, all instances of C_1 (instances in the both areas A_1 and A_2) will be returned successfully. However, the instances in area A_3 of class C_3 will be returned as well. Similar situation applies to the retrieval problem of class C_2.

The problem presented above can be solved by introducing a concept of active functions/bits to different class-wise code-prototypes. In the example shown in Figure 1(a), if one defines w_1 and w_2 to be active functions and w_3 be inactive for class C_1 as shown in Figure 1(b), then given a query of class C_1 that falls in either area A_1 or area A_2, all instances in class C_1 not the other classes will be returned within Hamming radius 0. The case for class C_2 is similar if one defines w_1 and w_3 to be the two active functions for class C_2 as shown in Figure 1(c). Note that by introducing active hash functions to different class-wise code-prototypes, the storage of binary codes for training instances can be further saved as for each training instance, one only needs to store those bits that are "active" in the corresponding class-wise code-prototype. For example, for the instances in class C_1, only the 1st and 2nd bits are needed to be stored, and the 3rd bit can be discarded.

By summarizing the motivations mentioned above, we come up with a high-level idea on our hashing model design. In order to learn a precise and compact hash code for each instance, we aim to learn a code-prototype for each class, whose length is the same as the number of hash functions to be learned. The constructed code for each instance should be similar to its corresponding class-wise code-prototype. The class-wise code-prototypes should be able to capture the similarity between classes, and are associated with class-specific active bits such that searching is more effective and efficient. In the rest of this paper, we denote by Class-wise Supervised Hashing (CSH) our proposed learning to hash method.

2 Related Work

Our work is related to sparse hashing methods [Xia *et al.*, 2015; Masci *et al.*, 2013; Zhu *et al.*, 2013; Wu *et al.*, 2014; Zhang *et al.*, 2016b]. Among these methods, [Zhu *et al.*, 2013; Wu *et al.*, 2014; Zhang *et al.*, 2016b; 2016a] proposed to convert sparse coefficients into $0/1$ binary code or non-negative integers, which are used to represent the instances based on high-level bases or a dictionary. Such kind of sparse binary codes can be obtained by using $0/1$ instead of $-1/+1$ for each bit. In other words, the zero-valued bits in [Zhu *et al.*, 2013; Wu *et al.*, 2014; Zhang *et al.*, 2016b] are similar to those of value -1 in our method. We use $+1$ and -1 to construct binary code and 0 to indicate sparsity of the code. Masci *et al.* [2013] proposed to add sparsity constraints on the constructed hash codes, while Xia *et al.* [2015] and Zhang *et al.* [2016a] proposed to learn a sparse projection matrix for hash functions. These two approaches can both help saving storage space, but to different extents. In general, when the size of training instances is much larger than that of feature dimensions, which is common in many real-world applications, the former approach can save more storage space.

Different from [Masci *et al.*, 2013] to add sparsity constraints on the constructed hash codes for individual instances, our proposed method is able to generate sparse binary codes for labeled training instances through learning a set of class-wise code-prototypes with active bits for different classes. In addition, in most previous methods, the L_1 norm is used to induce sparsity, which is a relaxation of the L_0 norm, while in our proposed method, we used the L_0 norm to control the numbers of active bits for each class-wise code-prototype, and develop an effective algorithm to solve the resultant optimization problem. Note that the concept of active bits was first proposed in [Liu *et al.*, 2012b], where a hash model is learned by exploring the sparse association between hash bits and classes. However, it was specially designed for multi-label classification problems, and fails to be applied to multi-class classification problems as studied in this paper, where the co-occurrence information among classes is not available.

3 Methodology

Assume we are given n training labeled instances $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ..., (\mathbf{x}_n, y_n)\}$, where $\mathbf{x}_i \in \mathbb{R}^d$ is an input instance, and $y_i \in \{1, 2, ..., L\}$ is the corresponding class label. Without loss of generality, we assume the input instances to be zero centered, i.e., $\sum_{i=1}^{n}\mathbf{x}_i = 0$. A hash model aims to map each input instance $\mathbf{x}_i$ to a binary code of r bits through learning r hash functions as follows,

$$\mathbf{b}_i = [h_1(\mathbf{x}_i), h_2(\mathbf{x}_i), ..., h_r(\mathbf{x}_i)]^\top = \text{sgn}(\mathbf{W}^\top \mathbf{x}_i), \quad (1)$$

where each hash function $h_j(\mathbf{x})$ is associated with a weight vector $\mathbf{w}_j \in \mathbb{R}^r$, and $\mathbf{W} = [\mathbf{w}_1, \mathbf{w}_2, ..., \mathbf{w}_r] \in \mathbb{R}^{d\times r}$ is the hash projection matrix. The sgn$(\mathbf{x})$ function is an element-wise sign function defined as sgn$(x) = 1$ if $x \geq 0$, and -1 otherwise. $\mathbf{b}_i \in \{1, -1\}^r$ is the resultant r-bit hash code for $\mathbf{x}_i$.

3.1 An Overview of the Proposed Method

As discussed in Section 1, previous supervised or semi-supervised hashing methods aim to enforce the binary codes to be similar (or different) if the corresponding instances belong to the same class (or different classes). One typical approach is to maximize $\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbf{K}_{ij}\mathbf{b}_i^\top\mathbf{b}_j$ or minimize $\sum_{i=1}^{n}\sum_{j=1}^{n}\left(\mathbf{K}_{ij} - \frac{1}{r}\mathbf{b}_i^\top\mathbf{b}_j\right)^2$, where $\mathbf{K} \in \mathbb{R}^{n\times n}$ is the similarity matrix between instances. When n is large, the cost in terms of both storage and computation is expensive. Though sampling techniques can be applied to approximate the similarity matrix, some label information may be discarded.

To fully exploit all labeled instances to learn hash functions and scale up the learning process at the same time, we propose to learn an additional set of class-wise code-prototypes, denoted by $\mathbf{a}_j$ $(j = 1, 2, ..., L)$, based on a class-pairwise similarity matrix $\mathbf{S} \in \mathbb{R}^{L\times L}$, where $\mathbf{a}_j$ is a corresponding r-bit binary code for class $j \in \{1, 2, ..., L\}$, and $\mathbf{S}_{jk}$ is positive if classes j and k are similar, otherwise $\mathbf{S}_{jk}$ is negative. Regarding the similarity measure between classes, we can simply define the similarity between class j and itself to be 1, and the similarity between different classes to be -1, i.e., $\mathbf{S}_{jk} = 1$ if $j = k$, otherwise, $\mathbf{S}_{jk} = -1$. When the classes lie in a taxonomy, we can encode the taxonomy structure to construct the class-pairwise similarity matrix. Specifically, motivated by [Weinberger and Chapelle, 2008], we first denote by $\mathbf{C}$ a cost matrix with $\mathbf{C}_{jk} \geq 0$ being the length of the shortest path between class j and class k in the taxonomy, where $j, k \in \{1, .., L\}$, and the path is defined to be 0 if $j = k$. We then normalize the cost matrix by $\bar{\mathbf{C}} = \mathbf{D}^{-1/2}\mathbf{C}\mathbf{D}^{-1/2}$, where $\mathbf{D}_{jj} = \sum_k \mathbf{C}_{jk}$ is a diagonal matrix. Finally, we define the similarity $\mathbf{S}$ with $\mathbf{S}_{jk} = -\bar{\mathbf{C}}_{jk}$ if $j \neq k$, otherwise, $\mathbf{S}_{jk} = 1$.

We expect the learned class-wise code-prototypes to have the following properties:

1. Based on the similarity matrix between classes, similar classes should have similar code-prototypes, while dissimilar classes should have different code-prototypes. This can be done by maximizing $\sum_{j=1}^{L}\sum_{k=1}^{L}\mathbf{S}_{jk}\mathbf{a}_j^\top\mathbf{a}_k$
2. Each class-wise code-prototype $\mathbf{a}_j$ should be able to guide the learning of hash codes for the instances of class j. This can be done by maximizing $\sum_{i=1}^{n}\mathbf{a}_{y_i}^\top\mathbf{b}_i$, where $y_i \in \{1, 2, ..., L\}$ is the corresponding class label of $\mathbf{b}_i$ or $\mathbf{x}_i$.
3. Each class-wise prototype-code $\mathbf{a}_j$ should have its specific active bits. To indicate active or inactive bits, we require $\mathbf{a}_j$ to be in $\{-1, +1, 0\}^r$. The l-th bit in $\mathbf{a}_j$ is active if its value is nonsparse, i.e., $\mathbf{a}_{j[l]} = -1$ or $\mathbf{a}_{j[l]} = +1$, otherwise, inactive.

3.2 Objective Function

By formulating our high-level ideas described above, we come up with the following objective function,

$$\max_{\mathbf{W},\mathbf{b}_i,\mathbf{a}_j} \sum_{i=1}^{n}\mathbf{a}_{y_i}^\top\mathbf{b}_i + \alpha\sum_{j=1}^{L}\sum_{k=1}^{L}\mathbf{S}_{jk}\mathbf{a}_j^\top\mathbf{a}_k - \frac{\beta}{2}\sum_{i=1}^{n}\left\|\mathbf{b}_i - \mathbf{W}^\top\mathbf{x}_i\right\|_2^2,$$
$$s.t. \quad \mathbf{b}_i \in \{1, -1\}^r, \ \mathbf{a}_j \in \{1, 0, -1\}^r, \ \|\mathbf{a}_j\|_0 \leq m. \quad (2)$$

where $\mathbf{b}_i$ is a binary code for $\mathbf{x}_i$, $i = 1, ..., n$, while $\mathbf{a}_j$ is a sparse class-wise code-prototype for class $j \in \{1, ..., L\}$, $\mathbf{a}_{y_i}$ is the corresponding class-wise code-prototype of $\mathbf{x}_i$, $\mathbf{S}$ is the class-pairwise similarity matrix, $\mathbf{W}$ is the hash projection matrix, α and β are positive tradeoff parameters, $\|\cdot\|_0$ is the zero-norm that returns the number of nonsparse elements, and m is the parameter controlling the sparsity of each $\mathbf{a}_j$.

In the objective of (2), the first term is to enforce $\mathbf{a}_{y_i}$ and $\mathbf{b}_i$ to be close to each other. The second term is to ensure that if classes j and k are similar (or dissimilar), the corresponding class-wise code-prototypes $\mathbf{a}_j$ and $\mathbf{a}_k$ are similar (or dissimilar). The third term is the negative of a loss function on binary fitting errors, which is commonly used in many learning to hash methods [Gong and Lazebnik, 2011; Shen *et al.*, 2015; Pan *et al.*, 2015; Xia *et al.*, 2015]. The constraint $\|\mathbf{a}_j\|_0 \leq m$ is to enforce each $\mathbf{a}_j$ to be of at most m active bits.

4 Optimization

To find a feasible solution for the optimization problem (2), in this section, we present an alternating optimization approach.

4.1 W-Step: fix $\mathbf{b}_i$ and $\mathbf{a}_j$, update $\mathbf{W}$

By fixing $\mathbf{b}_i$ and $\mathbf{a}_j$, the optimization problem (2) becomes

$$\min_{\mathbf{W}} \sum_{i=1}^{n}\left\|\mathbf{b}_i - \mathbf{W}^\top\mathbf{x}_i\right\|_2^2. \quad (3)$$

By defining $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n]$ and $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_n]$, the optimization problem (3) can be further rewritten as a regression problem

$$\min_{\mathbf{W}} \left\|\mathbf{B} - \mathbf{W}^\top\mathbf{X}\right\|_F^2,$$

for which we have the solution,

$$\mathbf{W} = (\mathbf{X}\mathbf{X}^\top)^{-1}\mathbf{X}\mathbf{B}^\top. \quad (4)$$

4.2 B-Step: fix $\mathbf{W}$ and $\mathbf{a}_j$, update $\mathbf{b}_i$

By fixing $\mathbf{W}$ and $\mathbf{a}_j$, which means $\mathbf{a}_{y_i}$ is fixed for each $\mathbf{x}_i$, and defining a matrix $\hat{\mathbf{A}} \in \{1, 0, -1\}^{r\times n}$ with the i-th column corresponding to the class prototype $\mathbf{a}_{y_i}$ of $\mathbf{x}_i$, the optimization problem (2) can be reformulated as

$$\max_{\mathbf{B}} \quad Q(\mathbf{B}) \quad (5)$$
$$s.t. \quad \mathbf{B} \in \{1, -1\}^{r\times n},$$

where $Q(\mathbf{B}) = \text{tr}(\hat{\mathbf{A}}^\top\mathbf{B}) - \frac{\beta}{2}\left\|\mathbf{B} - \mathbf{W}^\top\mathbf{X}\right\|_F^2$. To solve this optimization problem, we further rewrite $Q(\mathbf{B})$ as follows,

$$Q(\mathbf{B}) = \text{tr}(\hat{\mathbf{A}}^\top\mathbf{B}) - \frac{\beta}{2}\left(\|\mathbf{B}\|_F^2 + \|\mathbf{W}^\top\mathbf{X}\|_F^2 - 2\text{tr}(\mathbf{X}^\top\mathbf{W}\mathbf{B})\right).$$

Since $\|\mathbf{B}\|_F^2$ and $\|\mathbf{W}^\top\mathbf{X}\|_F^2$ are both constant,

$$\begin{aligned} Q(\mathbf{B}) &= \mathrm{tr}(\hat{\mathbf{A}}^\top\mathbf{B}) + \beta\mathrm{tr}(\mathbf{X}^\top\mathbf{W}\mathbf{B}) + \mathrm{const} \\ &= tr(\mathbf{V}^\top\mathbf{B}) + \mathrm{const}, \qquad (6) \end{aligned}$$

where $\mathbf{V} = \hat{\mathbf{A}} + \beta\mathbf{W}^\top\mathbf{X}$. Maximizing $Q(\mathbf{B})$ is equivalent to maximizing $\mathrm{tr}(\mathbf{V}^\top\mathbf{B})$. As $\mathbf{B} \in \{1,-1\}^{r\times n}$, the optimal solution for (5) can be obtained by setting

$$\mathbf{B} = \mathrm{sgn}(\hat{\mathbf{A}} + \beta\mathbf{W}^\top\mathbf{X}). \qquad (7)$$

4.3 A-Step: fix W and $\mathbf{b}_i$, update $\mathbf{a}_j$

Finally, by fixing $\mathbf{W}$ and $\mathbf{b}_i$, the optimization problem (2) can be rewritten as follows,

$$\begin{aligned} \max_{\mathbf{a}_j} \quad & \sum_{i=1}^{n} \mathbf{a}_{y_i}^\top \mathbf{b}_i + \alpha \sum_{j=1}^{L}\sum_{k=1}^{L} \mathbf{S}_{jk}\mathbf{a}_j^\top\mathbf{a}_k, \qquad (8) \\ s.t. \quad & \mathbf{a}_j \in \{1,0,-1\}^r,\ \|\mathbf{a}_j\|_0 \le m. \end{aligned}$$

Note that the expression $\mathbf{a}_{y_i}$ is not consistent with $\mathbf{a}_j$. Thus, we need to transform it to $\mathbf{a}_j$ for convenience in solution induction. To do this, we define $\widetilde{\mathbf{b}}_j$ as the sum of $\mathbf{b}_i$'s which belong the j-th class, i.e., $\widetilde{\mathbf{b}}_j = \sum_{i\in\mathcal{C}} \mathbf{b}_i$, where $\mathcal{C} = \{i|y_i = j\ \&\ i = 1,2,...,n\}$. Then (8) can be rewritten as

$$\begin{aligned} \max_{\mathbf{a}_j} \quad & \sum_{j=1}^{L} \mathbf{a}_j^\top \widetilde{\mathbf{b}}_j + \alpha \sum_{j=1}^{L}\sum_{k=1}^{L} \mathbf{S}_{jk}\mathbf{a}_j^\top\mathbf{a}_k, \qquad (9) \\ s.t. \quad & \mathbf{a}_j \in \{1,0,-1\}^r,\ \|\mathbf{a}_j\|_0 \le m. \end{aligned}$$

However, the optimization problem (9) is still difficult to optimize because both the zero-norm constraint (or sparsity constraint) and the discrete constraint on $\mathbf{a}_j$ are not differentiable. Furthermore, the second term of the objective in (8) is a quadratic term with respect to $\mathbf{a}_j$, making it difficult to obtain solutions for all $\mathbf{a}_j$ at the same time by using the optimization method used in (5).

Alternatively, we propose an algorithm to update only one $\mathbf{a}_j$ at each time. To avoid confusion, we denoted by $\mathbf{a}_t$ the one to be optimized at each time, and $\mathbf{a}_j$, where $j \neq t$, any one of the remaining constant vectors. Let $\widetilde{\mathbf{s}}$ be a column vector corresponding to the t-th row of $\mathbf{S}$ except for $\mathbf{S}_{tt}$, i.e., $\widetilde{\mathbf{s}} = [\mathbf{S}_{t1}, \mathbf{S}_{t2}, ..., \mathbf{S}_{t,t-1}, \mathbf{S}_{t,t+1}, ..., \mathbf{S}_{tL}]^\top$, $\widetilde{\mathbf{A}} = [\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_{t-1}, \mathbf{a}_{t+1}, ..., \mathbf{a}_L]$, and $\mathcal{K} = \{1,2,...,L\} \setminus \{t\}$. When all $\{\mathbf{a}_j\}$'s except for $\mathbf{a}_t$ are fixed, we can derive that

$$\sum_{j=1}^{L} \mathbf{a}_j^\top \widetilde{\mathbf{b}}_j = \mathbf{a}_t^\top \widetilde{\mathbf{b}}_t + \mathrm{const} = \widetilde{\mathbf{b}}_t^\top \mathbf{a}_t + \mathrm{const}, \qquad (10)$$

and

$$\begin{aligned} & \sum_{j=1}^{L}\sum_{k=1}^{L} \mathbf{S}_{jk}\mathbf{a}_j^\top\mathbf{a}_k \\ &= \sum_{j\in\mathcal{K}}\sum_{k\in\mathcal{K}} \mathbf{S}_{jk}\mathbf{a}_j^\top\mathbf{a}_k + \sum_{j\in\mathcal{K}} \mathbf{S}_{jt}\mathbf{a}_j^\top\mathbf{a}_t + \sum_{j\in\mathcal{K}} \mathbf{S}_{tj}\mathbf{a}_t^\top\mathbf{a}_j + \mathbf{S}_{tt}\mathbf{a}_t^\top\mathbf{a}_t \\ &= 2\widetilde{\mathbf{s}}^\top\widetilde{\mathbf{A}}^\top\mathbf{a}_t + \mathbf{S}_{tt}\mathbf{a}_t^\top\mathbf{a}_t + \mathrm{const}. \qquad (11) \end{aligned}$$

By substituting (10) and (11) into (9), we obtain the optimization problem with respect to $\mathbf{a}_t$ as follows,

$$\begin{aligned} \max_{\mathbf{a}_t} \quad & \widetilde{\mathbf{b}}_t^\top\mathbf{a}_t + 2\alpha\widetilde{\mathbf{s}}^\top\widetilde{\mathbf{A}}^\top\mathbf{a}_t + \alpha\mathbf{S}_{tt}\mathbf{a}_t^\top\mathbf{a}_t \qquad (12) \\ s.t. \quad & \mathbf{a}_t \in \{1,0,-1\}^r,\ \|\mathbf{a}_t\|_0 \le m. \end{aligned}$$

Proposition 1. *To solve the maximization problem (12), the number of non-zero elements of $\mathbf{a}_t$ should be maximal. In other words, $\|\mathbf{a}_t\|_0 = m$.*

Proof. Define a feasible solution for (12) as $\mathbf{a}_p$, where $\|\mathbf{a}_p\|_0 = v \le m-1$, and another feasible solution as $\mathbf{a}_q$, where $\|\mathbf{a}_q\|_0 = v+1$. Assume that these two solutions or vectors are different only on the l-th element, where $\mathbf{a}_{p[l]} = 0$ and $\mathbf{a}_{q[l]} \neq 0$. Let $Z(\mathbf{a}) = \widetilde{\mathbf{b}}_t^\top\mathbf{a} + 2\alpha\widetilde{\mathbf{s}}^\top\widetilde{\mathbf{A}}^\top\mathbf{a} + \alpha\mathbf{S}_{tt}\mathbf{a}^\top\mathbf{a}$, and $\mathbf{u} = \widetilde{\mathbf{b}}_t + 2\alpha\widetilde{\mathbf{A}}\widetilde{\mathbf{s}}$. Then we have

$$Z(\mathbf{a}_q) - Z(\mathbf{a}_p) = \mathbf{u}_{[l]}(\mathbf{a}_{q[l]} - \mathbf{a}_{p[l]}) + v + 1 - v = \mathbf{u}_{[l]}\mathbf{a}_{q[l]} + 1.$$

If $\mathbf{a}_{q[l]} = \mathrm{sgn}(\mathbf{u}_{[l]})$, then $Z(\mathbf{a}_q) - Z(\mathbf{a}_p) > 1$. This means that for any feasible solution $\mathbf{a}_p$ whose zero norm is less than m, we can always find another feasible solution $\mathbf{a}_q$ such that $Z(\mathbf{a}_q) > Z(\mathbf{a}_p)$. Therefore, the optimal solution for (12) is not $\mathbf{a}_p$ but the one whose zero norm equals to m. □

With Proposition 1, the third term of the objective in (12) equals to a constant $\alpha \times m$. Thus, the objective becomes to maximize $\widetilde{\mathbf{b}}_t^\top\mathbf{a}_t + 2\alpha\widetilde{\mathbf{s}}^\top\widetilde{\mathbf{A}}^\top\mathbf{a}_t$ or $\mathbf{u}^\top\mathbf{a}_t$ by using the definition of $\mathbf{u}$. It can be shown that the optimal solution is

$$\mathbf{a}_{t[l]} = \begin{cases} \mathrm{sgn}(\mathbf{u}_{[l]}), & \text{if } |\mathbf{u}_{[l]}| \ge \eta, \\ 0, & \text{otherwise}, \end{cases} \qquad (13)$$

where $|\mathbf{u}_{[l]}|$ denotes the absolute value of $\mathbf{u}_{[l]}$, and η is the m-th largest absolute value of the elements of $\mathbf{u}$. We can use (13) to obtain solutions for all $\{\mathbf{a}_j\}$'s.

Note that for initialization, we set each class-wise code-prototype $\mathbf{a}_j$ to be a vector of all zeros. To accelerate the convergence, we initialize the projection matrix $\mathbf{W}$ using random sampling from a uniform distribution $\mathbf{E}(0)$ as used in LSH.

4.4 Search in Sparse Binary Codes

As discussed in previous sections, the binary codes for instances, $\{\mathbf{b}_i\}$'s, are not constrained to be sparse in training, while the class-wise code-prototypes, $\{\mathbf{a}_j\}$'s, are learned with sparsity constraints. After $\{\mathbf{a}_j\}$'s are learned, for any binary code $\mathbf{b}_i$ of $\mathbf{x}_i$ in the database, if its label is known, we can convert it into a sparse version by setting its inactive bits to 0 based on its corresponding class-wise code-prototype $\mathbf{a}_{y_i}$.

Note that for search on a database of n non-sparse binary codes of r bits within Hamming radius R, it needs to go through all the hash buckets within R bits of the query binary code [Norouzi *et al.*, 2012]. The number of buckets to be examined is $\sum_{i=0}^{R}\binom{r}{i}$, and therefore this costs $O(r!R!)$. In our proposed method, as for each instance, only m active bits needs to be considered, the time complexity for R Hamming radius neighbors search is $O(m!R!)$. If R is not zero, the active bits strategy used in the proposed method will save much retrieval time.

5 Experiment

The datasets we employ to test the performance of the proposed CSH method include the Animals with Attributes dataset[2] (AwA) and the CIFAR-100 dataset [Krizhevsky,

[2] http://attributes.kyb.tuebingen.mpg.de/

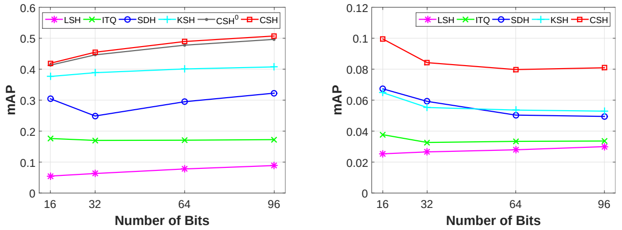
2009]. **AwA** consists of 30,475 images of 50 classes of animals, which provides six kinds of pre-extracted feature representations. In our experiments, we use the 4096-dimensional DECAF features, which are obtained via a 7-layer CaffeNet. There is no taxonomy information among classes. However, as all classes are mammal animals, we use part of the Mammals subtrees of WordNet as the taxonomy to construct the class-pairwise similarity matrix. On this dataset, 5% of data (i.e. $1,457$ instances) are randomly picked up as queries, and the remained instances form a training set. **CIFAR-100** consists of 100 classes with each class containing 600 color images of size 32×32, which results in 60K images in total. In experiments, every image is represented by 512-dimensional GIST features. And 58K instances are randomly selected from the whole set to comprise a training set while the remained 2K instances are used as queries. As only 2-layer taxonomy is provided, in our experiment, we do not encode the taxonomy information into the the similarity matrix for this dataset.

5.1 Methods for Comparison

We compare our proposed CSH method with four hashing methods. They are Locality-Sensitive Hashing (LSH) [Charikar, 2002], Iterative Quantization (ITQ) [Gong and Lazebnik, 2011], Supervised Discrete Hashing (SDH) [Shen *et al.*, 2015] and Supervised Hashing with Kernel (KSH) [Liu *et al.*, 2012a]. The model of LSH is obtained through random sampling. Its performance is guaranteed by probability theory. ITQ is an unsupervised learning method, whose promising performance has been shown compared with other unsupervised hashing algorithms. Therefore, we consider it as a representative for unsupervised hashing. SDH is a recently proposed supervised hashing with a discrete constraint. The discrete constraint is also used in CSH. Though SDH is designed for binary classification, it has shown its effectiveness in ANN search. Moreover, the optimization approach used in SDH is similar to ours. Hence, we choose it as a comparison method. KSH is a famous supervised hashing method based on an instance-pairwise similarity matrix. As the out-of-space issue caused by KSH is the one we aim address in this paper. Therefore, we choose KSH for conducting comparison experiments as well.

5.2 Experimental Setup

In the optimization problem (2) for CSH, there are three parameter α, β and m. The tradeoff parameters α and β are to balance the impact of three terms in the objective. The first term is to sum up n items, the second term is to sum up $L \times L$ items, and the third term is to sum up n items. Suppose each item be equally important in the objective. We set $\alpha = n/L^2$ and $\beta = n/n = 1$. Regarding the sparsity parameter m, we set $m = \frac{3}{4}r$, where r is the code length. Sensitivity study on different degree of sparsity is also conducted through experiments. For initialization on $\mathbf{W}$ in CSH, we adopt the same strategy as LSH. To avoid bias in initialization, we run CSH and LSH 10 times with different random initializations on $\mathbf{W}$[3], and report the averaged results.



(a) AwA (b) CIFAR-100

Figure 2: mAP v.s. code length.

Regarding other comparison methods, as "anchors" are required for both SDH and KSH. To avoid bias in anchors selection, we randomly sample 10 different anchor sets from the training data, each of which consists of $1,000$ anchors. We run SDH and KSH with the 10 anchor sets, and report the averaged results. In addition, though it has been shown that with only 5% of training data, KSH can still perform well [Liu *et al.*, 2012a], in our experiments, we feed KSH as many training instances as possible for generating the similarity matrix to maximize its performance in terms of search quality. With the computational resources we have, we can generate an instance-pairwise similarity matrix of around $30K$ instances. Therefore, for KSH, we use all training instances of AwA and $30K$ training instances of CIFAR-100. Other parameters for the comparison methods are set based on the suggested values reported in the original papers.

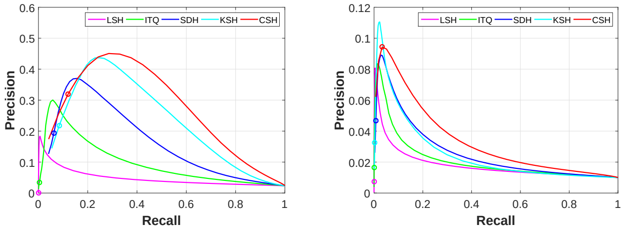
5.3 Evaluation Criteria

The mean Average Precision (mAP) is the most popular evaluation criterion for hashing methods [Wang *et al.*, 2010b; Liu *et al.*, 2012a; 2014; Wang *et al.*, 2010a; Gong and Lazebnik, 2011; Shen *et al.*, 2015; Xia *et al.*, 2015; Wu *et al.*, 2013]. Therefore, we adopt it as one of evaluation criteria. However, we find that with binary representations, a lot of instances may have the same Hamming distance to a given query, and thus have the same ranking positions. In this case, mAP scores may be misleading. Therefore, we also use the precision and recall curve within different Hamming radii to a given query to evaluate the hashing performance.

5.4 Comparison Results

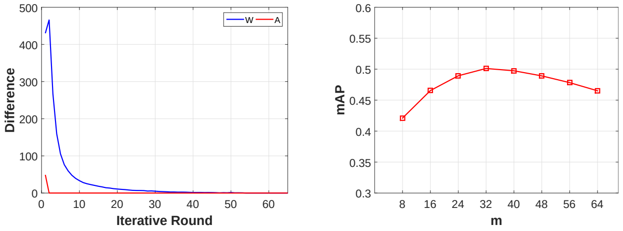
We first compare CSH with the other four hashing methods in terms of mAP in Figure 2. Clearly, CSH outperforms the other baselines with different code lengths on AwA as shown in Figure 2(a). We observe that the three supervised hashing methods, CSH, SDH, and KSH, perform better than the unsupervised methods, ITQ and LSH. This indicate the importance of supervised information for semantic search. Note that CSH without encoding class taxonomy information is also tested on AwA by setting all diagonal entries of the similarity matrix $\mathbf{S}$ to be $+1$, and all the other entries to be -1, which is denoted by CSH^0 in the figure. The result reveals that CSH can outperform all other competitors, no matter supervised or unsupervised ones, even without engaging the

[3] In each run, CSH and LSH share the same initialized $\mathbf{W}$.



(a) AwA (b) CIFAR-100

Figure 3: Precision v.s. Recall.



(a) Convergence (b) Impact of m

Figure 4: Properties verification

class taxonomy into learning. The performance of CSH is slightly better than CSH^0, which implies that encoding class taxonomy can help to generate a more precise class-pairwise similarity matrix, and thus improve search quality.

We also find that the results on CIFAR-100 in terms of mAP are misleading. To be specific, in Figure 2(b), the mAP scores achieved by all the hashing methods are much lower than those shown in Figure 2(a). This implies that learning hash functions on CIFAR-100 is much difficult than that on AwA. In this case, the code of small length, e.g., code of 16 bits, is supposed to perform poor. However, from the figure, the hashing functions with the smallest code length perform best. This is because for a difficult task, with small length of code, a lot of binary codes have the same Hamming distance to a given query. In this case, mAP scores are not reliable.

Therefore, we generate the precision-recall curve in Figure 3, where the code length is of 64 bits, and the precision and recall within Hamming radius 2 (PH2 and RH2) are highlighted using circles. In general, the performance is considered to be better if the area under the precision-recall curve is larger. Apparently, CSH performs best on both AwA and CIFAR-100 in terms of the area under the precision-recall curve. Note that the precision-recall curve shown in Figure 3 is quite different from that for information retrieval or classification problems, where the precision would be very high when the recall is close to zero. Here, the precision is low when the recall is close to zero. This phenomenon is reasonable in hashing methods because the Hamming distance between the query and a retrieved instance is discrete, resulting in lots of retrieved instances having the same ranking position. If no retrieved instances have the same code as the query, both the precision and recall are zero, which is the case for LSH and ITQ in our experiments. When retrieved instances have the same code as the query, as some of them may belong to a different class from the query, the precision may not be high, which is the case for KSH, SDH and CSH.

More specifically, as can be seen from results on AwA shown in Figure 3(a), the precision-recall curve of CSH covers largest area. The ranking of the five comparison methods in terms of the area under precision-recall curve is the same as that in terms of mAP. Furthermore, PH2 and RH2 of CSH are much better than the other four methods. From the results on CIFAR-100 shown in Figure 3(b), the precision-recall curve of CSH also covers largest area. Though the precision of KSH is better than CSH when the recall is very low, the gap is not large. Moreover, PH2 and RH2 of CSH are much better than those of the other methods. Except for CSH, RH2s of the other four methods are very close to zero, which means that the codes generated by the these methods on CIFAR-100 are not as compact as CSH. This therefore proves the effectiveness of sparse binary codes constructed by CSH.

5.5 Convergence and Effect of Sparsity Parameter

We also conduct experiments on convergence analysis and impact study on the sparsity degree of each $\mathbf{a}_i$ for CSH. The experiments are done on AwA with the code length fixed to be 64 bits. The convergence analysis on the projection matrix $\mathbf{W}$ and the class-wise code-prototypes matrix $\mathbf{A}$ is shown in Figure 4(a), where the y axis is the difference in Frobenius norm between two successive iterations. As can be seen from the figure, $\mathbf{A}$ converges within 5 iterations, and $\mathbf{W}$ begins to converge after 10 iterations, and stays unchanged after 40 iterations. The impact of the sparsity degree on each $\mathbf{a}_i$, i.e., the value of m in (2), to the final hashing performance is shown in Figure 4(b), where m varies from 8 to 64 with 8 increment each step. From the figure, we observe that when m is very small, such as $m = 8$ and $m = 16$, the mAP score is comparatively low because the number of the active bits may not be sufficient enough to capture important information. When m becomes larger, such as $m = 56$ or $m = 64$, the mAP scores drop as most binary codes become "active". This reveals the importance of inactivating a certain number of bits to improve the performance, which is illustrated by Figure 1. These experimental results suggest us to choose the value of m in the range of $[\frac{3}{8}r, \frac{3}{4}r]$, where r is the length of a code.

6 Conclusion

In this paper we propose a novel class-wise supervised hashing method, denoted by CSH. Instead of using an instance-pairwise similarity matrix, CSH is based on a class-pairwise similarity matrix. With the class-pairwise similarity matrix, a set of class-wise code-prototypes with class-specific active binary bits are introduced to help generating more compact hashing functions, and thus enable more effective and efficient search. Experimental results verify the superiority of CSH in terms of search quality and storage space over other baseline hashing methods.

Acknowledgments

This work is supported by the NTU Singapore Nanyang Assistant Professorship (NAP) grant M4081532.020.

References

[Charikar, 2002] Moses Charikar. Similarity estimation techniques from rounding algorithms. In *ACM Symposium on Theory of Computing*, pages 380–388, 2002.

[Gionis *et al.*, 1999] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via hashing. In *International Conference on Very Large Data Bases*, pages 518–529, 1999.

[Gong and Lazebnik, 2011] Yunchao Gong and Svetlana Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 817–824, 2011.

[Krizhevsky, 2009] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[Lin *et al.*, 2014] Guosheng Lin, Chunhua Shen, Qinfeng Shi, Anton van den Hengel, and David Suter. Fast supervised hashing with decision trees for high-dimensional data. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1971–1978, 2014.

[Liu *et al.*, 2011] Wei Liu, Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Hashing with graphs. In *International Conference on Machine Learning*, pages 1–8, 2011.

[Liu *et al.*, 2012a] Wei Liu, Jun Wang, Rongrong Ji, Yu-Gang Jiang, and Shih-Fu Chang. Supervised hashing with kernels. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 2074–2081, 2012.

[Liu *et al.*, 2012b] Xianglong Liu, Yadong Mu, Bo Lang, and Shih-Fu Chang. Compact hashing for mixed image-keyword query over multi-label images. In *International Conference on Multimedia Retrieval*, page 18, 2012.

[Liu *et al.*, 2014] Wei Liu, Cun Mu, Sanjiv Kumar, and Shih-Fu Chang. Discrete graph hashing. In *Neural Information Processing Systems*, pages 3419–3427, 2014.

[Masci *et al.*, 2013] Jonathan Masci, Alexander M. Bronstein, Michael M. Bronstein, Pablo Sprechmann, and Guillermo Sapiro. Sparse similarity-preserving hashing. *CoRR*, 2013.

[Norouzi *et al.*, 2012] Mohammad Norouzi, Ali Punjani, and David J. Fleet. Fast search in hamming space with multi-index hashing. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3108–3115, 2012.

[Pan *et al.*, 2015] Yingwei Pan, Ting Yao, Houqiang Li, Chong-Wah Ngo, and Tao Mei. Semi-supervised hashing with semantic confidence for large scale visual search. In *ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 53–62, 2015.

[Shen *et al.*, 2015] Fumin Shen, Chunhua Shen, Wei Liu, and Heng Tao Shen. Supervised discrete hashing. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 37–45, 2015.

[Wang *et al.*, 2010a] Jun Wang, Ondrej Kumar, and Shih-Fu Chang. Semi-supervised hashing for scalable image retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3424–3431, 2010.

[Wang *et al.*, 2010b] Jun Wang, Sanjiv Kumar, and Shih-Fu Chang. Sequential projection learning for hashing with compact codes. In *International Conference on Machine Learning*, pages 1127–1134, 2010.

[Wang *et al.*, 2014] Jingdong Wang, Heng Tao Shen, Jingkuan Song, and Jianqiu Ji. Hashing for similarity search: A survey. *CoRR*, abs/1408.2927, 2014.

[Wang *et al.*, 2016] Jun Wang, Wei Liu, Sanjiv Kumar, and Shih-Fu Chang. Learning to hash for indexing big data - A survey. *Proceedings of the IEEE*, 104(1):34–57, 2016.

[Weinberger and Chapelle, 2008] Kilian Q. Weinberger and Olivier Chapelle. Large margin taxonomy embedding for document categorization. In *Conference on Neural Information Processing Systems*, pages 1737–1744, 2008.

[Weiss *et al.*, 2008] Yair Weiss, Antonio Torralba, and Robert Fergus. Spectral hashing. In *Neural Information Processing Systems*, pages 1753–1760, 2008.

[Wu *et al.*, 2013] Chenxia Wu, Jianke Zhu, Deng Cai, Chun Chen, and Jiajun Bu. Semi-supervised nonlinear hashing using bootstrap sequential projection learning. *IEEE Transactions on Knowledge and Data Engineering*, 25(6):1380–1393, 2013.

[Wu *et al.*, 2014] Fei Wu, Zhou Yu, Yi Yang, Siliang Tang, Yin Zhang, and Yueting Zhuang. Sparse multi-modal hashing. *IEEE Transactions on Multimedia*, 16(2):427–439, 2014.

[Xia *et al.*, 2015] Yan Xia, Kaiming He, Pushmeet Kohli, and Jian Sun. Sparse projections for high-dimensional binary codes. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 3332–3339, 2015.

[Zhang *et al.*, 2016a] Lihe Zhang, Huchuan Lu, Dandan Du, and Luning Liu. Sparse hashing tracking. *IEEE Transactions on Image Processing*, 25(2):840–849, 2016.

[Zhang *et al.*, 2016b] Yin Zhang, Weiming Lu, Yang Liu, and Fei Wu. Kernelized sparse hashing for scalable image retrieval. *Neurocomputing*, 172:207–214, 2016.

[Zhu *et al.*, 2013] Xiaofeng Zhu, Zi Huang, Hong Cheng, Jiangtao Cui, and Heng Tao Shen. Sparse hashing for fast multimedia search. *ACM Trans. Inf. Syst.*, 31(2):9:1–9:24, 2013.