Semi-Supervised Hashing for Large Scale Search

Jun Wang, *Member, IEEE,* Sanjiv Kumar, *Member, IEEE,* and Shih-Fu Chang, *Fellow, IEEE*

Abstract—Hashing based approximate nearest neighbor (ANN) search in huge databases has become popular owing to its computational and memory efficiency. The popular hashing methods, e.g., *Locality Sensitive Hashing* and *Spectral Hashing*, construct hash functions based on random or principal projections. The resulting hashes are either not very accurate or inefficient. Moreover these methods are designed for a given metric similarity. On the contrary, semantic similarity is usually given in terms of pairwise labels of samples. There exist supervised hashing methods that can handle such semantic similarity but they are prone to overfitting when labeled data is small or noisy. In this work, we propose a semi-supervised hashing (*SSH*) framework that minimizes empirical error over the labeled set and an information theoretic regularizer over both labeled and unlabeled set. Based on this framework, we present three different *semi-supervised hashing* methods, including orthogonal hashing, non-orthogonal hashing, and sequential hashing. Particularly, the sequential hashing method generates robust codes in which each hash function is designed to correct the errors made by the previous ones. We further show that the sequential learning paradigm can be extended to unsupervised domains where no labeled pairs are available. Extensive experiments on four large datasets (up to 80 million samples) demonstrate the superior performance of the proposed *SSH* methods over state-of-the-art supervised and unsupervised hashing techniques.

Index Terms—Hashing, Nearest neighbor search, binary codes, semi-supervised hashing, pairwise labels, sequential hashing.

1 Introduction

Web data including documents, images and videos is growing rapidly. For example, the photo sharing website Flickr has over 5 billion images. The video sharing website YouTube receives more than 24 hours of uploaded videos per minute. There is an emerging need to retrieve relevant content from such massive databases. Besides the widely-used text-based commercial search engines, like Google and Bing, content based image retrieval (CBIR) has attracted substantial attention over the past decade [1]. Instead of taking textual keywords as input, CBIR techniques directly take a visual query q and try to return its nearest neighbors from a given database $\mathcal{X}$ using a prespecified feature space and distance measure.

In fact, finding nearest neighbors is a fundamental step in many machine learning algorithms such as kernel density estimation, spectral clustering, manifold learning and semi-supervised learning [2]. Exhaustively comparing the query q with each sample in the database $\mathcal{X}$ is infeasible because linear complexity $\mathcal{O}(|\mathcal{X}|)$ is not scalable in practical settings. Besides the scalability issue, most large scale CBIR applications also suffer from *curse of dimensionality* since visual descriptors usually have hundreds or even thousands of dimensions. Therefore, beyond the infeasibility of exhaustive search, storage of the original data also becomes a critical bottleneck.

- *J. Wang is with IBM T.J. Watson Research, Yorktown Heights, NY, 10598. E-mail: wangjun@us.ibm.edu*
- *S. Kumar is with Google Research, New York, NY, 10011 E-mail: sanjivk@google.com*
- *S.-F. Chang is with the Department of Electrical and Computer Engineering, Columbia University, New York, NY, 10027. E-mail: sfchang@ee.columbia.edu.*

Fortunately, in many applications, it is sufficient to return approximate nearest neighbors. Instead of doing exact nearest neighbor search through linear scan, a fast and accurate indexing method with sublinear ($o(|\mathcal{X}|)$), logarithmic ($\mathcal{O}(\log|\mathcal{X}|)$), or even constant ($\mathcal{O}(1)$) query time is desired for approximate nearest neighbors (ANN) search. For example, ϵ-ANN aims at finding $p \in \mathcal{X}$ satisfying $d(p,q) \leq (1+\epsilon)d(p',q)$, where $\epsilon > 0$ and p' is the nearest neighbor of query q [3]. Over the past several decades, many techniques have been developed for fast and efficient ANN search. Especially, tree based approaches tend to store data with efficient data structures, which makes the search operation extremely fast, typically with the complexity of $\mathcal{O}(\log(|\mathcal{X}|))$. The representative tree based algorithms include KD tree [4][5][6], ball tree [7], metric tree [8], and vantage point tree [9]. A more detailed survey of the tree based ANN search algorithms can be found in [10]. However, the performance of tree-based methods degrades drastically for high-dimensional data, mostly reducing to worst case of linear search [11].

In addition, tree-based methods also suffer from memory constraints. In many cases, the size of the data-structure is bigger than the original data itself. Hence, hashing based ANN techniques have attracted more attention recently. They have constant query time and also need substantially reduced storage as they usually store only compact codes. In this work, we focus on binary codes. Given n D-dim vectors $\mathbf{X} \in \mathbb{R}^{D\times n}$, the goal in hashing is to learn suitable K-bit binary codes $\mathbf{Y} \in \mathbb{B}^{K\times n}$. To generate $\mathbf{Y}$, K binary hash functions are used. Linear projection-based hash functions have been widely used in the literature since they are very simple and efficient. Also, they have achieved state-of-the-art performance for various tasks [12][13][14]. In linear projection-based hashing, the k^{th}

hash function can be generalized to be of the following form:

$$h_k(\mathbf{x}) = \text{sgn}(f(\mathbf{w}_k^\top \mathbf{x} + b_k)), \quad (1)$$

where $\mathbf{x}$ is a data point, $\mathbf{w}_k$ is a projection vector, b_k is a threshold and $f(\cdot)$ is an arbitrary function. Since $h(\mathbf{x}) \in \{-1, 1\}$, the corresponding binary hash bit can be simply expressed as: $y_k(\mathbf{x}) = (1 + h_k(\mathbf{x}))/2$. Different choices of $\mathbf{w}$ and $f(\cdot)$ lead to different hashing approaches.

Broadly, hashing methods can be divided into two main categories: unsupervised methods and supervised methods. Unsupervised methods design hash functions using unlabeled data $\mathcal{X}$ to generate binary codes. Locality Sensitive Hashing (*LSH*) [12] is arguably the most popular unsupervised hashing method and has been applied to many problem domains, including information retrieval and computer vision. Its kernelized version has recently been developed in [15]. Another effective method called Spectral Hashing (*SH*) was proposed recently by Weiss et al. [13]. More recently, graph based hashing technique was proposed to leverage low-dimensional manifold structure of data to design efficient and compact hash codes [16]. Since unsupervised methods do not require any labeled data, they can be easily applied to different data domains given a prespecified distance metric.

From the perspective of quality of retrieved results, hashing based *ANN* methods aim to return an approximate set of nearest neighbors. However, in typical CBIR, even returning the exact nearest neighbors does not guarantee good search quality. This is due to the well-known problem of *semantic gap*, where the high level semantic description of visual content often differs from the extracted low level visual descriptors [17]. Furthermore, most hashing approaches provide theoretic guarantees with only certain distance metric spaces. For instance, *LSH* function family works for the ℓ_p ($p \in (0, 2]$) and Jaccard distances. But in CBIR applications, it is usually hard to express similarity (or distance) between image pairs with a simple metric. Ideally, one would like to provide pairs of images that one believes contain 'similar' or 'dissimilar' images. From such pairwise labeled data, the hashing mechanism should be able to automatically generate codes that satisfy the semantic similarity. Supervised learning techniques have been proposed in the past to handle this issue. For example, in [18], authors suggested merging standard *LSH* with a learned Mahalanobis metric to reflect semantic indexing. Since this approach uses labeled sample pairs for training distance metric, it was categorized as a semi-supervised learning paradigm. However, the hash functions are still randomized.

In addition, a Boosted Similarity Sensitive Coding (*BSSC*) technique was proposed in [19] which tries to learn a series of weighted hash functions from labeled data. Kulis and Darrell recently proposed to learn hash functions based on explicitly minimizing the reconstruction error between the metric space and Hamming space, termed as Binary Reconstructive Embedding (*BRE*) [21]. Other binary encoding methods, like deep neural network stacked with Restricted Boltzmann Machines (*RBMs*), were recently applied to learn binary codes [20], which have shown superior performance over *BSSC* given sufficient training labels [23]. Although *RBMs* use both labeled and unlabeled data, the latter is only used in a pre-training phase, whose solution provides a good initialization for the supervised back-propagation phase. So *RBMs* based binary embedding is still categorized as a supervised method. One of the problems with all of these supervised methods is that they have much slower training process in comparison to the unsupervised methods. Another problem stems from limited or noisy training data, which can easily lead to overfitting.

Realizing the challenging issue of *semantic gap*, and inefficiencies of existing supervised hashing approaches, in this article, we propose a *Semi-Supervised Hashing* (*SSH*) framework that can leverage semantic similarity using labeled data while remaining robust to overfitting. The objective function of *SSH* consists of two main components: supervised empirical fitness and unsupervised information theoretic regularization. Specifically, we provide a rigorous formulation in which a supervised term tries to minimize the empirical error on the labeled data while an unsupervised term provides effective regularization by maximizing desirable properties like variance and independence of individual bits. Based on this semi-supervised formulation, we present several variants of learning binary hash functions. The taxonomy of the proposed Semi-supervised hashing method in comparison to a few popular hashing methods is given in Table 1.

The remainder of this article is organized as follows. In Section 2, we briefly survey several popular hashing methods. Section 3 presents the detailed formulation of our approach, i.e. Semi-Supervised Hashing (*SSH*). In Section 4, we present three different solutions for designing semi-supervised hash functions, followed by an extension to unsupervised domain. Section 5 provides extensive experimental validation on several large image datasets. The conclusions and future work are given in Section 6.

2 Related Work

Given a dataset $\mathbf{X} = \{\mathbf{x}_i\}, i = 1, \cdots, n$, containing $n = |\mathbf{X}|$ points, and $\mathbf{x}_i \in \mathbb{R}^D$, the objective in nearest neighbor search is to find a set of nearest neighbors $\mathcal{R} \subset \mathbf{X}$ for a given query q. For large-scale applications, to avoid excessive computational and memory costs, one would like to instead do an *approximate nearest neighbor* (ANN) search with sublinear query complexity [2][12].

In the following subsections, in addition to the popularly used Locality Sensitive Hashing, we briefly review a few state-of-the-art representative methods from supervised as well as unsupervised domains. Specifically, we discuss Boosted Similarity Sensitive Coding, Spectral Hashing, and Binary Reconstructive Embedding based hashing along with their pros and cons for the application of image retrieval.

2.1 Locality Sensitive Hashing

A key ingredient of *Locality Sensitive Hashing* (*LSH*) is mapping "similar" samples to the same bucket with high probability. In other words, the property of locality in the original space will be largely preserved in the Hamming space. More precisely, the hash functions $h(\cdot)$ from *LSH* family satisfy the following elegant locality preserving property:

$$P\{h(\mathbf{x}) = h(\mathbf{y})\} = sim(\mathbf{x}, \mathbf{y}) \quad (2)$$

Method	Hash Function	Projection	Hamming Distance	Learning Paradigm
Locality Sensitive Hashing (LSH) [12]	$\mathrm{sgn}(\mathbf{w}^\top \mathbf{x} + b)$	data-independent	non-weighted	unsupervised
Shift Invariant Kernel based Hashing (SIKH) [14]	$\mathrm{sgn}(cos(\mathbf{w}^\top \mathbf{x} + b) + t)$	data-independent	non-weighted	unsupervised
Spectral Hashing (SH) [13]	$\mathrm{sgn}(cos(k\mathbf{w}^\top \mathbf{x}))$	data-dependent	non-weighted	unsupervised
Boosted Similarity Sensitive Coding (BSSC) [19]	–	data-dependent	weighted	supervised
Restricted Boltzmann Machines (RBMs) [20]	–	–	non-weighted	supervised
Binary Reconstructive Embedding (BRE) [21]	$\mathrm{sgn}(\mathbf{w}^\top \mathbf{k}_\mathbf{x})$	data-dependent	non-weighted	supervised
Label-regularized Max-margin Partition (LAMP) [22]	$\mathrm{sgn}(\mathbf{w}^\top \mathbf{x} + b)$	data-dependent	non-weighted	supervised
LSH for Learned Metrics (LSH-LM) [18]	$\mathrm{sgn}(\mathbf{w}^\top \mathbf{G}\mathbf{x})$	data-dependent	non-weighted	supervised
Semi-Supervised Hashing (SSH)	$\mathrm{sgn}(\mathbf{w}^\top \mathbf{x} + b)$	data-dependent	non-weighted	semi-supervised

TABLE 1
The conceptual comparison of the proposed *SSH* method with other binary encoding methods.

where the similarity measure can be directly linked to the distance function d, for example, $sim(\mathbf{x}, \mathbf{y}) = \exp\{-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{\sigma^2}\}$. A typical category of *LSH* functions consists of random projections and thresholds as:

$$h(x) = sign(\mathbf{w}^\top \mathbf{x} + b) \tag{3}$$

where $\mathbf{w}$ is a random hyperplane and b is a random intercept. Clearly, the random vector $\mathbf{w}$ is data-independent, which is usually constructed by sampling each component of $\mathbf{w}$ randomly from a p-stable distribution for a general ℓ_p metric, where $p \in (0, 2]$, e.g., standard Gaussian for ℓ_2 distance [24]. Due to the asymptotic theoretical guarantees for random projection based *LSH*, many *LSH* based large scale search systems have been developed. For instance, a self-tuning indexing technique, called *LSH* forest was proposed in [25], which aims at improving the performance without additional storage and query overhead. However, the practical efficiency of *LSH* is still very limited since it requires long codes to achieve high precision. But this reduces recall dramatically, and constructing multiple tables is necessary to increase recall [12]. For example, suppose a K-bit binary embedding is given as $H(x) = [h_1(x), \cdots, h_K(x)]$. Then, given l K-bit tables, the collision probability for two points is given as:

$$P\{H(\mathbf{x}) = H(\mathbf{y})\} \propto l \cdot \left[1 - \frac{cos^{-1}\mathbf{x}^\top \mathbf{y}}{\pi}\right]^K \tag{4}$$

For a large scale application, the value of K should be large to reduce false collisions (i.e., the number of non-neighbor sample pairs falling into the same bucket). However a large value of K decreases the collision probability between similar samples as well. In order to overcome this drawback, multiple hash tables have to be constructed. Obviously, this is inefficient due to extra storage cost and larger query time. In [26], a technique called MultiProbe *LSH* was developed to reduce the number of required hash tables through intelligently probing multiple buckets in each hash table. However, the above data-independent random projections based hash functions lack good discrimination over data. Therefore, recent methods tend to leverage data-dependent learning techniques to improve the efficiency of hash functions [27].

Incorporating kernel learning with *LSH* can help generalize similarity search from standard metric space to a wide class of similarity functions [15][22]. Furthermore, metric learning has been combined with randomized *LSH* functions given a few pairwise similarity and dissimilarity constraints [18]. All these methods still use random hyperplanes to design hash functions with asymptotic performance guarantees. However, in practice, the performance is significantly degraded if only compact codes are used [13][28].

2.2 Boosted Similarity Sensitive Coding

To improve discrimination among hash codes, Boosted Similarity Sensitive Coding (*BSSC*) was designed to learn a weighted Hamming embedding for task specific similarity search [19] as,

$$H : \mathcal{X} \rightarrow \{\alpha_1 h_1(\mathbf{x}), \cdots, \alpha_K h_K(\mathbf{x})\} \tag{5}$$

Hence the conventional Hamming distance is replaced by the weighted version as

$$d_{\mathcal{WH}} = \sum_{k=1}^{K} \alpha_k |h_k(\mathbf{x}_i) - h_k(\mathbf{x}_j)| \tag{6}$$

By learning the hash function weights $\{\alpha_1, \cdots, \alpha_k\}$, the objective is to lower the collision probability of non-neighbor pair $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}$, while improving the collision probability of neighborhood pair $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}$. If one treats each hash function as a decision stump, the straightforward way of learning the weights is to directly apply adaptive boosting algorithm [29], as described in [19].

2.3 Spectral Hashing

Due to the limitations of random projection based hashing approaches, learning techniques have been applied to improve the efficiency of hashing. Particularly, *Spectral Hashing* (*SH*) was recently proposed to design compact binary codes for ANN search. Besides the desired property of keeping neighbors in input space as neighbors in the Hamming space,

the basic *SH* formulation requires the codes to be balanced and uncorrelated. Strictly speaking, the hash functions $H(\mathbf{x}) = \{h_k(\mathbf{x})\}, k = 1, \cdots, K$ satisfy the following criteria [13]:

$$\min \sum_{ij} sim(\mathbf{x}_i, \mathbf{x}_j) \|H(\mathbf{x}_i) - H(\mathbf{x}_j)\|^2 \quad (7)$$

$$\text{subject to: } h_k(\mathbf{x}_i) \in \{-1, 1\}$$
$$\sum_i h_k(\mathbf{x}_i) = 0, k = 1, \cdots, K$$
$$\sum_i h_k(\mathbf{x}_i) h_l(\mathbf{x}_i) = 0, \text{for } k \neq l$$

The direct solution for the above optimization is non-trivial for even a single bit since it is a balanced graph partition problem, which is NP hard. The combination of K-bit balanced partitioning is even harder because of the pairwise independence constraints. After relaxing the constraints, the above optimization was solved using spectral graph analysis [30]. Especially, with the assumption of uniform data distribution, the spectral solution can be efficiently computed using 1D-Laplacian eigenfunctions [13].

The final *SH* algorithm consists of three key steps: 1) extraction of maximum variance directions through Principal Component Analysis (PCA) on the data; 2) direction selection, which prefers to partition projections with large spread and small spatial frequency; 3) partition of projected data by a sinusoidal function with previously computed angular frequency. *SH* has been shown to be effective in encoding low-dimensional data since the important PCA directions are selected multiple times to create binary bits. However, for high dimensional problems ($D >> K$) where many directions contain enough variance, usually each PCA direction is picked only once. This is because the top few projections have similar range and thus, a low spatial frequency is preferred. In this case, *SH* approximately replicates a PCA projection followed by a mean partition. In *SH*, the projection directions are data dependent but learned in an unsupervised manner. Moreover, the assumption of uniform data distribution is usually not true for real-world data.

2.4 Binary Reconstructive Embedding

Instead of using data-independent random projections as in *LSH* or principal components as in *SH*, Kulis and Darrell [21] proposed data-dependent and bit-correlated hash functions as:

$$h_k(\mathbf{x}) = \text{sgn}\left(\sum_{q=1}^{s} \mathbf{W}_{kq} \kappa(\mathbf{x}_{kq}, \mathbf{x})\right) \quad (8)$$

The sample set $\{\mathbf{x}_{kq}\}, q = 1, \cdots, s$ is the training data for learning hash function h_k and $\kappa(\cdot)$ is a kernel function, and $\mathbf{W}$ is a weight matrix.

Based on the above formulation, a method called Binary Reconstructive Embedding (*BRE*) was designed to minimize a cost function measuring the difference between the metric and reconstructed distance in Hamming space. The Euclidean metric $d_{\mathcal{M}}$ and the binary reconstruction distance $d_{\mathcal{R}}$ are defined as:

$$d_{\mathcal{M}}(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{2}\|\mathbf{x}_i - \mathbf{x}_j\|^2 \quad (9)$$
$$d_{\mathcal{R}}(\mathbf{x}_i, \mathbf{x}_j) = \frac{1}{K}\sum_{k=1}^{K} (h_k(\mathbf{x}_i) - h_k(\mathbf{x}_j))^2$$

The objective is to minimize the following reconstruction error to derive the optimal $\mathbf{W}$:

$$\mathbf{W}^* = \arg\min_{\mathbf{W}} \sum_{(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{N}} [d_{\mathcal{M}}(\mathbf{x}_i, \mathbf{x}_j) - d_{\mathcal{R}}(\mathbf{x}_i, \mathbf{x}_j)]^2 \quad (10)$$

where the set of sample pairs $\mathcal{N}$ is the training data. Optimizing the above objective function is difficult due to the non-differentiability of $\text{sgn}(\cdot)$ function. Instead, a coordinate-descent algorithm was applied to iteratively update the hash functions to a local optimum. This hashing method can be easily extended to a supervised scenario by setting same-label pairs to have zero distance and different-label pairs to have a large distance. However, since the binary reconstruction distance $d_{\mathcal{R}}$ is bounded in $[0, 1]$ while the metric distance $d_{\mathcal{M}}$ has no upper bound, the minimization problem in Eq. (10) is only meaningful when input data is appropriately normalized. In practice, the original data point $\mathbf{x}$ is often mapped to a hypersphere with unit length so that $0 \leq d_{\mathcal{M}} \leq 1$. This normalization removes the scale of data points, which is often not negligible in practical applications of nearest neighbor search.

3 Semi-Supervised Paradigm for Hashing

In this section, we present the formulation of our hashing method, i.e. *Semi-Supervised Hashing* (*SSH*). In this setting, one is given a set of n points, $\mathcal{X} = \{\mathbf{x}_i\}$, $i = 1 \ldots n$, $\mathbf{x}_i \in \mathbb{R}^D$, in which a fraction of pairs are associated with two categories of label information, $\mathcal{M}$ and $\mathcal{C}$. Specifically, a pair $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}$ is denoted as a neighbor-pair when $(\mathbf{x}_i, \mathbf{x}_j)$ are neighbors in a metric space or share common class labels. Similarly, $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}$ is called a nonneighbor-pair if two samples are far away in metric space or have different class labels. Let us denote the data matrix by $\mathbf{X} \in \mathbb{R}^{D \times n}$ where each column is a data point. Also, suppose there are l points, $l \ll n$, which are associated with at least one of the categories $\mathcal{M}$ or $\mathcal{C}$. Let us denote the matrix formed by these l columns of $\mathbf{X}$ as $\mathbf{X}_l \in \mathbb{R}^{D \times l}$. The goal of *SSH* is to learn hash functions that minimize the error on the labeled training data $\mathbf{X}_l$, while maximally satisfying the desirable properties of hashing e.g., maximizing information from each bit. We start the discussion on our learning paradigm with the basic formulation of *SSH*.

3.1 Empirical Fitness

SSH aims to map the data $\mathbf{X} \in \mathbb{R}^{D \times n}$ to a Hamming space to obtain its compact representation. Suppose we want to learn K hash functions leading to a K-bit Hamming embedding of $\mathbf{X}$ given by $\mathbf{Y} \in \mathbb{B}^{K \times n}$. Without loss of generality, let $\mathbf{X}$ be normalized to have zero mean. In this work, we use linear projection coupled with mean thresholding as a hash function.

In other words, given a vector $\mathbf{w}_k \in \mathbb{R}^D$, the k^{th} hash function is defined as,

$$h_k(\mathbf{x}_i) = \text{sgn}(\mathbf{w}_k^\top \mathbf{x}_i + b_k) \tag{11}$$

where b_k is the mean of the projected data, i.e., $b_k = -\frac{1}{n}\sum_{j=1}^{n} \mathbf{w}_k^\top \mathbf{x}_j = 0$ since $\mathbf{X}$ is zero-mean. One can get the corresponding binary bit as,

$$y_{ki} = \frac{1}{2}(1 + h_k(\mathbf{x}_i)) = \frac{1}{2}(1 + \text{sgn}(\mathbf{w}_k^\top \mathbf{x}_i)) \tag{12}$$

Let $\mathbf{H} = [h_1, \ldots, h_K]$ be a sequence of K hash functions and $\mathbf{W} = [\mathbf{w}_1, \ldots, \mathbf{w}_K] \in \mathbb{R}^{D\times K}$. We want to learn a $\mathbf{W}$ that gives the same bits for $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}$ and different bits for $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}$. An objective function measuring the empirical accuracy on the labeled data for a family of hashing functions $\mathbf{H}$ can be defined as:

$$J(\mathbf{H}) = \sum_k \left\{ \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{M}} h_k(\mathbf{x}_i)h_k(\mathbf{x}_j) - \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{C}} h_k(\mathbf{x}_i)h_k(\mathbf{x}_j) \right\} \tag{13}$$

One can express the above objective function in a compact matrix form by first defining a matrix $\mathbf{S} \in \mathbb{R}^{l\times l}$ incorporating the pairwise labeled information from $\mathbf{X}_l$ as:

$$S_{ij} = \begin{cases} 1 & : \ (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M} \\ -1 & : \ (\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C} \\ 0 & : \ otherwise. \end{cases} \tag{14}$$

Also, suppose $\mathbf{H}(\mathbf{X}_l) \in \mathbb{B}^{K\times l}$ maps the points in $\mathbf{X}_l$ to their K-bit hash codes. Then, the objective function $J(\mathbf{H})$ can be represented as,

$$\begin{aligned} J(\mathbf{H}) &= \frac{1}{2}\mathbf{tr}\left\{\mathbf{H}(\mathbf{X}_l)\ \mathbf{S}\ \mathbf{H}(\mathbf{X}_l)^\top\right\} \\ &= \frac{1}{2}\mathbf{tr}\left\{\text{sgn}(\mathbf{W}^\top\mathbf{X}_l)\ \mathbf{S}\ \text{sgn}(\mathbf{W}^\top\mathbf{X}_l)^\top\right\} \end{aligned} \tag{15}$$

where $\text{sgn}(\mathbf{W}^\top\mathbf{X}_l)$ is the matrix of signs of individual elements. In summary, we intend to learn optimal hashing functions $\mathbf{H}$ by maximizing the objective function as:

$$\mathbf{H}^* = \arg\max_{\mathbf{H}} J(\mathbf{H}) \tag{16}$$

Since the objective function $J(\mathbf{H})$ itself is non-differentiable, the above problem is difficult to solve even without considering any regularizer. We first present a simple relaxation of the empirical fitness.

In the relaxed version of the objective function, we replace the sign of projection with its *signed magnitude* in Eq. (13). This relaxation is quite intuitive in the sense that it not only desires similar points to have the same sign but also large projection magnitudes, meanwhile projecting dissimilar points not only with different signs but also as far as possible. With this relaxation, the new objective can be directly written as a function of $\mathbf{W}$ as,

$$J(\mathbf{W}) = \sum_k \left\{ \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{M}} \mathbf{w}_k^\top \mathbf{x}_i\mathbf{x}_j^\top \mathbf{w}_k - \sum_{(\mathbf{x}_i,\mathbf{x}_j)\in\mathcal{C}} \mathbf{w}_k^\top \mathbf{x}_i\mathbf{x}_j^\top \mathbf{w}_k \right\} \tag{17}$$

Without loss of generality, we also assume $\|\mathbf{w}_k\| = 1, \quad \forall k$. The above function can be expressed in a matrix form as

$$J(\mathbf{W}) = \frac{1}{2}\mathbf{tr}\left\{\mathbf{W}^\top\mathbf{X}_l\mathbf{S}\mathbf{X}_l^\top\mathbf{W}\right\}. \tag{18}$$

3.2 Information Theoretic Regularization

Maximizing empirical accuracy for just a few pairs can lead to severe overfitting, as illustrated in Figure 1. To get better generalization ability, one needs to add regularization by incorporating conditions that lead to desirable properties of hash codes. Even though empirical fitness uses only labeled data, the regularization term uses all the data X, both unlabeled and labeled, leading to a semi-supervised learning paradigm. Hence, we use a regularizer which utilizes both labeled and unlabeled data. From the information-theoretic point of view, one would like to maximize the information provided by each bit [31]. Using maximum entropy principle, a binary bit that gives balanced partitioning of $\mathbf{X}$ provides maximum information. Thus, it is desired to have $\sum_{i=1}^{n} h_k(\mathbf{x}_i) = 0$. However, finding mean-thresholded hash functions that meet the balancing requirement is hard. Instead, we use this property to construct a regularizer for the empirical accuracy given in Eq. (18). We now show that maximum entropy partitioning is equivalent to maximizing the variance of a bit.

Proposition 3.1. *[maximum variance condition] A hash function with maximum entropy $H(h_k(\mathbf{x}))$ must maximize the variance of the hash values, and vice-versa, i.e.,*

$$\max H(h_k(\mathbf{x})) \Longleftrightarrow \max \text{var}[h(\mathbf{x})]$$

Proof: Assume h_k has a probability p of assigning the hash value $h_k(\mathbf{x}) = 1$ to a data point and $1-p$ for $h_k(\mathbf{x}) = -1$. The entropy of $h_k(\mathbf{x})$ can be computed as

$$H(h_k(\mathbf{x})) = -p\log_2 p - (1-p)\log_2(1-p)$$

It is easy to show that the maximum entropy is $\max H(h_k(\mathbf{x})) = \mathbf{1}$ when the partition is balanced, i.e., $p = 1/2$. Now we show that balanced partitioning implies maximum bit variance. The mean of hash value is $E[h(\mathbf{x})] = \mu = 2p - 1$ and the variance is:

$$\begin{aligned} \text{var}[h_k(\mathbf{x})] &= E[(h_k(\mathbf{x}) - \mu)^2] \\ &= 4(1-p)^2 p + 4p^2(1-p) = 4p(1-p) \end{aligned}$$

Clearly, $\text{var}[h(\mathbf{x})]$ is concave with respect to p and its maximum is reached at $p = 1/2$, i.e. balanced partitioning. Also, since $\text{var}[h(\mathbf{x})]$ has a unique maximum, it is easy to see that the maximum variance partitioning also maximizes the entropy of the hash function. □

Using the above proposition, the regularizer term is defined as,

$$R(\mathbf{W}) = \sum_k \text{var}[h_k(\mathbf{x})] = \sum_k \text{var}[\text{sgn}(\mathbf{w}_k^\top \mathbf{x})] \tag{19}$$

Maximizing the above function with respect to $\mathbf{W}$ is still hard due to its non-differentiability. To overcome this problem, we now show that the maximum variance of a hash function is lower-bounded by the scaled variance of the projected data.

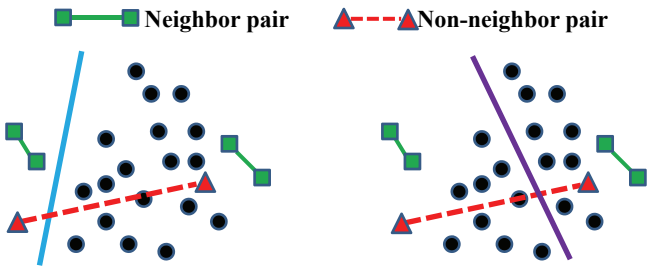


Fig. 1. An illustration of partitioning with maximum empirical fitness and entropy. Both of the partitions satisfy the given pairwise labels, while the right one is more informative due to higher entropy.

Proposition 3.2. *[lower bound on maximum variance of a hash function] The maximum variance of a hash function is lower-bounded by the scaled variance of the projected data, i.e.,*

$$\max \text{var}[h_k(\mathbf{x})] \geq \alpha \cdot \text{var}[\mathbf{w}_k^\top \mathbf{x}],$$

where α is a positive constant.

Proof: Suppose, $\|\mathbf{x}_i\|^2 \leq \beta \ \forall i, \beta > 0$. Since $\|\mathbf{w}_k\|^2 = 1 \ \forall k$, from Cauchy-Schwarz inequality,

$$\begin{aligned}
&\|\mathbf{w}_k^\top \mathbf{x}\|^2 \leq \|\mathbf{w}_k\|^2 \cdot \|\mathbf{x}\|^2 \leq \beta = \beta \cdot \|\operatorname{sgn}(\mathbf{w}_k^\top \mathbf{x})\|^2 \\
&\Rightarrow E\left[\|\operatorname{sgn}(\mathbf{w}_k^\top \mathbf{x})\|^2\right] \geq \frac{1}{\beta} E\left[\|\mathbf{w}_k^\top \mathbf{x}\|^2\right] \\
&\Rightarrow \max \text{var}[h_k(\mathbf{x})] \geq \frac{1}{\beta} \text{var}[\mathbf{w}_k^\top \mathbf{x}]
\end{aligned}$$

Here, we have used the properties that the data is zero-centered, i.e., $E[\mathbf{w}_k^\top \mathbf{x}] = 0$, and for maximum bit variance $E[\operatorname{sgn}(\mathbf{w}_k^\top \mathbf{x})] = 0$. □

Given the above proposition, we use the lower bound on the maximum variance of a hash function as a regularizer, which is easy to optimize, i.e.,

$$\begin{aligned}
R(\mathbf{W}) &= \frac{1}{\beta} \sum_k E[\|\mathbf{w}_k^\top \mathbf{x}\|^2] = \frac{1}{n\beta} \sum_k \mathbf{w}_k^\top \mathbf{X}\mathbf{X}^\top \mathbf{w}_k \\
&= \frac{1}{n\beta} \mathbf{tr}\left[\mathbf{W}^\top \mathbf{X}\mathbf{X}^\top \mathbf{W}\right] \qquad (20)
\end{aligned}$$

3.3 Final Objective Function

Combining the relaxed empirical fitness term from Eq. (18) and the relaxed regularization term from Eq. (20), the overall semi-supervised objective function is given as,

$$\begin{aligned}
J(\mathbf{W}) &= \frac{1}{2} \mathbf{tr}\left[\mathbf{W}^\top \mathbf{X}_l \mathbf{S} \mathbf{X}_l^\top \mathbf{W}\right] + \frac{\eta}{2} \mathbf{tr}\left[\mathbf{W}^\top \mathbf{X}\mathbf{X}^\top \mathbf{W}\right] \\
&= \frac{1}{2} \mathbf{tr}\{\mathbf{W}^\top \left[\mathbf{X}_l \mathbf{S} \mathbf{X}_l^\top + \eta \mathbf{X}\mathbf{X}^\top\right] \mathbf{W}\} \\
&= \frac{1}{2} \mathbf{tr}\{\mathbf{W}^\top \mathbf{M} \mathbf{W}\} \qquad (21)
\end{aligned}$$

where the constants n and β are absorbed in the coefficient η, and

$$\mathbf{M} = \mathbf{X}_l \mathbf{S} \mathbf{X}_l^\top + \eta \mathbf{X}\mathbf{X}^\top \qquad (22)$$

We refer to matrix $\mathbf{M}$ as *adjusted* covariance matrix. It is interesting to note the form of $\mathbf{M}$, where the unsupervised data variance part $\mathbf{X}\mathbf{X}^T$ gets adjusted by $\mathbf{X}_l \mathbf{S} \mathbf{X}_l^\top \mathbf{W}$ arising from the pairwise labeled data.

4 Projection Learning

4.1 Orthogonal Projection Learning

While learning compact codes, in addition to each bit being highly informative, one would like to avoid redundancy in bits as much as possible. One way to achieve this is by making the projection directions orthogonal, i.e.,

$$\begin{aligned}
\mathbf{W}^* &= \arg\max_{\mathbf{W}} J(\mathbf{W}) \qquad (23) \\
\text{subject to} \quad & \mathbf{W}^\top \mathbf{W} = \mathbf{I}
\end{aligned}$$

Now, the learning of optimal projections $\mathbf{W}$ becomes a typical eigenproblem, which can be easily solved by doing an eigenvalue decomposition on matrix $\mathbf{M}$:

$$\begin{aligned}
\max_{\mathbf{W}} J(\mathbf{W}) &= \sum_{k=1}^{K} \lambda_k \\
\mathbf{W}^* &= [\mathbf{e}_1 \cdots \mathbf{e}_K] \qquad (24)
\end{aligned}$$

where $\lambda_1 > \lambda_2 > \cdots > \lambda_K$ are the top eigenvalues of $\mathbf{M}$ and $\mathbf{e}_k, k = 1, \cdots, K$ are the corresponding eigenvectors.

Mathematically, it is very similar to finding maximum variance direction using PCA except that the original covariance matrix gets "adjusted" by another matrix arising from the labeled data. Hence, our framework provides an intuitive and easy way to learn hash functions in a semi-supervised paradigm.

4.2 Non-Orthogonal Projection Learning

In the previous subsection, we imposed orthogonality constraints on the projection directions in order to approximately decorrelate the hash bits. However, these orthogonality constraints sometimes lead to a practical problem. It is well known that for most real-world datasets, most of the variance is contained in top few projections. The orthogonality constraints force one to progressively pick those directions that have very low variance, substantially reducing the quality of lower bits, and hence the whole embedding. We empirically verify this behavior in Section 5. Depending on the application, it may make sense to pick a direction that is not necessarily orthogonal to the previous directions but has high variance as well as low empirical error on the labeled set. On the other hand, one doesn't want to pick a previous direction again since the fixed thresholds will generate the same hash codes in our case. Hence, instead of imposing hard orthogonality constraints, we convert them into a penalty term added to the objective function. This allows the learning algorithm to pick suitable directions by balancing various terms. With this, one can write the new objective function as,

$$\begin{aligned}
J(\mathbf{W}) &= \frac{1}{2} \mathbf{tr}\{\mathbf{W}^\top \mathbf{M} \mathbf{W}\} - \frac{\rho}{2} \|\mathbf{W}^\top \mathbf{W} - \mathbf{I}\|_{\mathcal{F}}^2 \qquad (25) \\
&= \frac{1}{2} \mathbf{tr}\{\mathbf{W}^\top \mathbf{M} \mathbf{W}\} - \frac{\rho}{2} \mathbf{tr}\left[(\mathbf{W}^\top \mathbf{W} - \mathbf{I})^\top (\mathbf{W}^\top \mathbf{W} - \mathbf{I})\right].
\end{aligned}$$

The new formulation has certain tolerance to non-orthogonality, which is modulated by a positive coefficient ρ. However, the above objective function is non-convex and there is no easy way to find the global solution unlike the previous case. To maximize the objective function J with respect to $\mathbf{W}$, we set the derivative to zero and absorb all constants into ρ as

$$\frac{\partial J(\mathbf{W})}{\partial \mathbf{W}} = 0 \Rightarrow (\mathbf{W}\mathbf{W}^\top - \mathbf{I} - \frac{1}{\rho}\mathbf{M})\mathbf{W} = 0 \tag{26}$$

Though the above equation admits infinite number of solutions since $\mathbf{W}$ has a non-empty nullspace, we can obtain a solution by ensuring

$$\mathbf{W}\mathbf{W}^\top\mathbf{W} = \left(\mathbf{I} + \frac{1}{\rho}\mathbf{M}\right)\mathbf{W}. \tag{27}$$

One can get a simple solution for the above condition if $\mathbf{I} + \frac{1}{\rho}\mathbf{M}$ is positive definite. From (21), $\mathbf{M}$ is symmetric but not necessarily positive definite. Let $\mathbf{Q} = \mathbf{I} + \frac{1}{\rho}\mathbf{M}$. Clearly, $\mathbf{Q}$ is also symmetric. In the following proposition we show that $\mathbf{Q}$ is positive definite if the coefficient ρ is chosen appropriately.

Proposition 4.1. *The matrix* $\mathbf{Q}$ *is positive definite if* $\rho > \max(0, -\bar{\lambda}_{min})$*, where* $\bar{\lambda}_{min}$ *is the smallest eigenvalue of* $\mathbf{M}$*.*

Proof: By definition in (25), $\rho > 0$. Since $\mathbf{M}$ is symmetric, it can be represented as $\mathbf{M} = \mathbf{U}diag(\lambda_1, \cdots, \lambda_D)\mathbf{U}^\top$ where all λ_i's are real. Let $\bar{\lambda}_{min} = \min(\lambda_1, \cdots, \lambda_D)$. Then $\mathbf{Q}$ can be written as

$$\begin{aligned}\mathbf{Q} &= \mathbf{I} + \mathbf{U}diag\left(\frac{\lambda_1}{\rho}, \cdots, \frac{\lambda_D}{\rho}\right)\mathbf{U}^\top \\ &= \mathbf{U}diag\left(\frac{\lambda_1}{\rho} + 1, \cdots, \frac{\lambda_D}{\rho} + 1\right)\mathbf{U}^\top\end{aligned}$$

Clearly, $\mathbf{Q}$ will have all eigenvalues positive if $\frac{\lambda_{min}}{\rho} + 1 > 0 \Rightarrow \rho > -\lambda_{min}$. □

If $\mathbf{Q}$ is positive definite, it can be decomposed as $\mathbf{Q} = \mathbf{L}\mathbf{L}^\top$ using Cholesky decomposition. Then, one can easily verify that $\mathbf{W} = \mathbf{L}\mathbf{U}$ satisfies Eq. (27). To achieve a meaningful *approximate* solution to our problem, we truncate the computed matrix $\mathbf{W}$ by selecting its first k columns. The final non-orthogonal projections are derived as,

$$\mathbf{W}_{\text{nonorth}} = \mathbf{L}\mathbf{U}_k \tag{28}$$

where $\mathbf{U}_k$ are the top k eigenvectors of $\mathbf{M}$.

4.3 Sequential Projection Learning

The above non-orthogonal solution is achieved by adjusting the previous orthogonal solution in a single step. However, this is one of many possible solutions which tend to work well in practice. One potential issue is that the above non-orthogonal solution is sensitive to the choice of the penalty coefficient ρ. To address these concerns, we further propose an alternative solution to learn a sequence of projections, which implicitly incorporates bit correlation by iteratively updating the pairwise label matrix. In addition, this iterative solution has the sequential error correction property where each hash function tries to correct the errors made by the previous one.

Algorithm 1 Sequential projection learning for hashing (*SPLH*)

Input: data $\mathbf{X}$, pairwise labeled data $\mathbf{X}_l$, initial pairwise labels $\mathbf{S}_1$, length of hash codes K, constant α
for $k = 1$ **to** K **do**
  Compute adjusted covariance matrix:
    $\mathbf{M}_k = \mathbf{X}_l\mathbf{S}_k\mathbf{X}_l^\top + \eta\mathbf{X}\mathbf{X}^\top$
  Extract the first eigenvector $\mathbf{e}$ of $\mathbf{M}_k$ and set:
    $\mathbf{w}_k = \mathbf{e}$
  Update the labels from vector $\mathbf{w}_k$:
    $\mathbf{S}_{k+1} = \mathbf{S}_k - \alpha\mathrm{T}\left(\tilde{\mathbf{S}}^k, \mathbf{S}_k\right)$
  Compute the residual:
    $\mathbf{X} = \mathbf{X} - \mathbf{w}_k\mathbf{w}_k^\top\mathbf{X}$
end for

The idea of sequential projection learning is quite intuitive. The hash functions are learned iteratively such that at each iteration, the pairwise label matrix $\mathbf{S}$ in (14) is updated by imposing higher weights on point pairs violated by the previous hash function. This sequential process implicitly creates dependency between bits and progressively minimizes empirical error. The sign of $\mathbf{S}_{ij}$, representing the logical relationship in a point pair $(\mathbf{x}_i, \mathbf{x}_j)$, remains unchanged in the entire process and only its magnitude $|\mathbf{S}_{ij}|$ is updated. Algorithm 1 describes the procedure of the proposed semi-supervised sequential projection learning method.

Suppose, $\tilde{\mathbf{S}}^k \in \mathbb{R}^{l\times l}$ measures the *signed magnitude* of pairwise relationships of the k^{th} projections of $\mathbf{X}_l$:

$$\tilde{\mathbf{S}}^k = \mathbf{X}_l^\top\mathbf{w}_k\mathbf{w}_k^\top\mathbf{X}_l \tag{29}$$

Mathematically, $\tilde{\mathbf{S}}^k$ is simply the derivative of empirical accuracy of k^{th} hash function, i.e., $\tilde{\mathbf{S}}^k = \nabla_{\mathbf{S}}J_k$, where $J_k = \mathbf{w}_k^\top\mathbf{X}_l\mathbf{S}\mathbf{X}_l^\top\mathbf{w}_k$. The function $\mathrm{T}(\cdot)$ implies the truncated gradient of J_k:

$$\mathrm{T}(\tilde{\mathbf{S}}_{ij}^k, \mathbf{S}_{ij}) = \begin{cases} \tilde{\mathbf{S}}_{ij}^k & : \ \mathrm{sgn}(\mathbf{S}_{ij}\cdot\tilde{\mathbf{S}}_{ij}^k) < 0 \\ 0 & : \ \mathrm{sgn}(\mathbf{S}_{ij}\cdot\tilde{\mathbf{S}}_{ij}^k) \geq 0 \end{cases} \tag{30}$$

The condition $\mathrm{sgn}(\mathbf{S}_{ij}\cdot\tilde{\mathbf{S}}_{ij}^k) < 0$ for a labeled pair $(\mathbf{x}_i, \mathbf{x}_j)$ indicates that hash bits $h_k(\mathbf{x}_i)$ and $h_k(\mathbf{x}_j)$ contradict the given pairwise label. In other words, points in a neighbor pair $(\mathbf{x}_i, \mathbf{x}_i) \in \mathcal{M}$ are assigned different bits or those in $(\mathbf{x}_i, \mathbf{x}_i) \in \mathcal{C}$ are assigned the same bit. For each such violation, $\mathbf{S}_{ij}$ is updated as $\mathbf{S}_{ij} = \mathbf{S}_{ij} - \alpha\tilde{\mathbf{S}}_{ij}^k$. The step size α is chosen such that $\alpha \leq \frac{1}{\beta}$ where $\beta = \max_i \|\mathbf{x}_i\|^2$, ensuring $|\alpha\tilde{\mathbf{S}}_{ij}^k| \leq 1$. This leads to numerically stable updates without changing the sign of $\mathbf{S}_{ij}$. Those pairs for which current hash function produces the correct bits, i.e., $\mathrm{sgn}(\mathbf{S}_{ij}\cdot\tilde{\mathbf{S}}_{ij}^k) > 0$, $\mathbf{S}_{ij}$ is kept unchanged by setting $\mathrm{T}(\tilde{\mathbf{S}}_{ij}^k, \mathbf{S}_{ij}) = 0$. Thus, those labeled pairs for which the current hash function does not predict the bits correctly exert more influence on the learning of the next function, biasing the new projection to produce correct bits for such pairs. Intuitively, it has a flavor of boosting-based methods commonly used for classification. However, unlike the standard boosting framework used for supervised learning, such as the one used in [19], the proposed method performs the

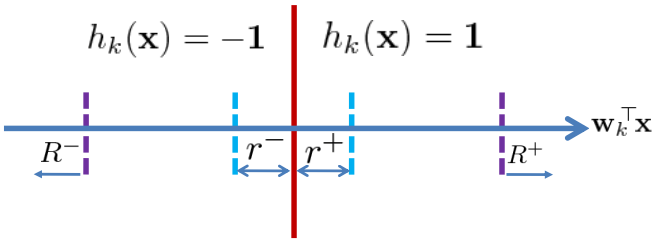


Fig. 2. Potential errors due to thresholding (red line) of the projected data to generate a bit. Points in r^- and r^+, are assigned different bits even though they are quite close. Also, points in R^- (R^+) and r^- (r^+) are assigned the same bit even though they are quite far.

sequential learning in a semi-supervised scenario, where the objective is to maximize the accuracy of the binary partition while maintaining the balancing of the partition. Furthermore, during the indexing and search process, the derived hash functions are treated equally instead of being weighted as in the boosting method.

After extracting a projection direction using $\mathbf{M}_k$, the contribution of the subspace spanned by that direction is removed from $\mathbf{X}$ to minimize the redundancy in bits. Note that this is not the same as imposing the orthogonality constraints on $\mathbf{W}$ discussed earlier. Since the supervised term $\mathbf{X}_l\mathbf{S}_k\mathbf{X}_l^\top$ still contains information potentially from the whole space spanned by original $\mathbf{X}$, the new direction may still have a component in the subspace spanned by the previous directions. Thus, the proposed formulation automatically decides the level of desired correlations between successive hash functions. If empirical accuracy is not affected, it prefers to pick uncorrelated projections. Unlike the non-orthogonal solution discussed in Section 4.2, the proposed sequential method aggregates various desirable properties in a single formulation leading to superior performance on real-world tasks as shown in Section 5. In fact, one can extend this sequential learning method in unsupervised cases as well, as shown in the next subsection.

4.4 Unsupervised Sequential Projection Learning

Unlike the semi-supervised case, pairwise labels are not available in the unsupervised case. To apply the general framework of sequential projection learning to an unsupervised setting, we propose the idea of generating pseudo labels at each iteration of learning. While generating a bit via a binary hash function, there are two types of boundary errors one encounters due to thresholding of the projected data. Suppose all the data points are projected on a one-dimensional axis as shown in Figure 2, and the red vertical line is the partition boundary, i.e. $\mathbf{w}_k^\top \mathbf{x} = 0$. The points left to the boundary are assigned a hash value $h_k(\mathbf{x}) = -1$ and those on the right are assigned a value $h_k(\mathbf{x}) = 1$. The regions marked as r^-, r^+ are located very close to the boundary and regions R^-, R^+ are located far from it. Due to thresholding, points in the pair $(\mathbf{x}_i, \mathbf{x}_j)$, where $\mathbf{x}_i \in r^-$ and $\mathbf{x}_j \in r^+$, are assigned different hash bits even though their projections are quite close. On the other hand, points in pair $(\mathbf{x}_i, \mathbf{x}_j)$, where $\mathbf{x}_i \in r^-$ and $\mathbf{x}_j \in R^-$ or $\mathbf{x}_i \in r^+$ and $\mathbf{x}_i \in R^+$, are assigned the same hash bit even though their projected values are quite far apart. To correct these two types of boundary "errors", we first introduce a neighbor-pair set $\mathcal{M}$ and a non-neighbor-pair set $\mathcal{C}$:

$$\mathcal{M} = \{(x_i, x_j)\} : h(x_i) \cdot h(x_j) = -1, |\mathbf{w}^\top(\mathbf{x}_i - \mathbf{x}_j)| \leq \epsilon$$
$$\mathcal{C} = \{(x_i, x_j)\} : h(x_i) \cdot h(x_j) = 1, |\mathbf{w}^\top(\mathbf{x}_i - \mathbf{x}_j)| \geq \zeta \quad (31)$$

Then, given the current hash function, a desired number of point pairs are sampled from both $\mathcal{M}$ and $\mathcal{C}$. Suppose, $\mathbf{X}_{\mathcal{MC}}$ contains all the points that are part of at least one sampled pair. Using the labeled pairs and $\mathbf{X}_{\mathcal{MC}}$, a pairwise label matrix $\mathbf{S}^k_{\mathcal{MC}}$ is constructed similar to Eq. (14). In other words, for a pair of samples $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{M}$, a pseudo label $\mathbf{S}^k_{\mathcal{MC}} = 1$ is assigned while for those $(\mathbf{x}_i, \mathbf{x}_j) \in \mathcal{C}$, $\mathbf{S}^k_{\mathcal{MC}} = -1$ is assigned. In the next iteration, these pseudo labels enforce point pair in $\mathcal{M}$ to be assigned the same hash values and those in $\mathcal{C}$ different ones. Thus it sequentially tries to correct the potential errors made by the previous hash functions. Note that the above discussion is based on the assumption that the pairs sampled from the close regions of opposite sides of the boundary are potential neighbors. In general, when the splitting hyperplane passes through the dense regions of data distribution, this assumption will be met. But when the hyperplane passes through sparse regions, it may be violated. Moreover, the number of available pseudo pairs may be too small to learn the next hash function reliably.

Note that each hash function $h_k(\cdot)$ produces a pseudo label set $\mathbf{X}^k_{\mathcal{MC}}$ and the corresponding label matrix $\mathbf{S}^k_{\mathcal{MC}}$. The new label information is used to adjust the data covariance matrix in each iteration of sequential learning, similar to that for the semi-supervised case. However, the unsupervised setting does not have a boosting-like update of the label matrix unlike the semi-supervised case. Each iteration results in its own pseudo label matrix depending on the hash function. Hence, to learn a new projection, all the pairwise label matrices since the beginning are used but their contribution is decayed exponentially by a factor δ at each iteration. Note that one does not need to store these matrices explicitly since incremental update can be done at each iteration resulting in the same memory and time complexity as for the semi-supervised case.

Algorithm 2 Unsupervised sequential projection learning for hashing (*USPLH*)

Input: data $\mathbf{X}$, length of hashing codes K
Initialize $\mathbf{X}^0_{\mathcal{MC}} = \emptyset, \mathbf{S}^0_{\mathcal{MC}} = \mathbf{0}$.
for $k = 1$ **to** K **do**
  Compute adjusted covariance matrix:
    $$\mathbf{M}_k = \sum_{i=0}^{k-1} \delta^{k-i} \mathbf{X}^i_{\mathcal{MC}} \mathbf{S}^i_{\mathcal{MC}} {\mathbf{X}^i_{\mathcal{MC}}}^\top + \eta \mathbf{X}\mathbf{X}^\top$$
  Extract the first eigenvector $\mathbf{e}$ of $\mathbf{M}_k$ and set:
    $\mathbf{w}_k = \mathbf{e}$
  Generate pseudo labels from projection $\mathbf{w}_k$:
    Sample $\mathbf{X}^k_{\mathcal{MC}}$ and construct $\mathbf{S}^k_{\mathcal{MC}}$
  Compute the residual:
    $\mathbf{X} = \mathbf{X} - \mathbf{w}_k \mathbf{w}_k^\top \mathbf{X}$
end for

The detailed learning procedure is described in algorithm chart 2. Since there exist no pseudo labels at the beginning, the first vector $\mathbf{w}_1$ is just the first principal direction of the data. Then, each hash function is learned to satisfy the pseudo labels iteratively by adjusting the data covariance matrix, similar to the *SPLH* approach.

To summarize, besides the three different versions of semi-supervised hashing methods, i.e., the orthogonal solution SSH_{orth}, the non-orthogonal solution $SSH_{nonorth}$, and the sequential solution *SPLH*, we also proposed an unsupervised extension of the sequential learning method in this section, named as *USPLH*.

5 EXPERIMENTS

We evaluated all the three versions of the proposed semi-supervised hashing methods, including orthogonal solution SSH_{orth}, non-orthogonal solution $SSH_{nonorth}$ (with orthogonality constraint relaxed), and sequential solution *SPLH*, as well as the unsupervised extension (*USPLH*) on several benchmark datasets. Their performance is compared with other popular binary coding methods, including Locality Sensitive Hashing (*LSH*), Spectral Hashing (*SH*), Binary Reconstructive Embedding (*BRE*), Boosted Similarity Sensitive Coding (BSSC), and Shift Invariant Kernel based Hashing (*SIKH*). These methods cover both unsupervised and supervised categories. Previous works have shown that *SH* performs better than other binary encoding methods [13], such as Restricted Boltzmann Machines (*RBMs*) [20] and *BSSC* [19]. For both *SH* and *BRE*, we used the best setting reported in previous literature. For *LSH*, we randomly select projections from a Gaussian distribution with zero-mean and identity covariance and apply random partitioning to construct hash functions. In addition, for all the supervised and semi-supervised methods, a small set of labeled samples was used during training. For example, only 1000 labeled images are used in the experiments on CIFAR10 dataset and 2000 on Flickr image data. Finally, for the proposed approach, we used cross validation to determine some parameters, such as the weight coefficient η and the step size α.

In the following subsections, we first discuss our evaluation protocols, followed by brief description of the benchmark datasets. Finally, extensive experimental results and comparisons are presented.

5.1 Evaluation Protocols

To perform fair evaluation, we adopt two criteria commonly used in the literature:

1) **Hamming ranking**: All the points in the database are ranked according to their Hamming distance from the query and the desired neighbors are returned from the top of the ranked list. The complexity of Hamming ranking is linear even though it is very fast in practice.
2) **Hash lookup**: A lookup table is constructed using the database codes, and all the points in the buckets that fall within a small Hamming radius r of the query are returned. The complexity of the hash lookups is constant time.

Note that evaluations based on *Hamming ranking* and *hash lookup* focus on different characteristics of hashing techniques. For instance, *hash lookup* emphasizes more on the practical search speed. However, when using many hash bits and single hash table, the Hamming space becomes increasingly sparse; and very few samples fall within the Hamming radius r ($r = 2$ in our setting), resulting in many failed queries without returned data points. In this situation, *Hamming ranking* provides better quality measurement of the Hamming embedding, while neglecting the issue of the search speed. All the experiments were conducted using a *single* hash table with relatively compact codes (up to 64 bits for the largest image collection dataset with around 80 million points). The search results are evaluated based on whether the returned images and the query sample share the same semantic labels for supervised and semi-supervised tests. We use several metrics to measure the quantitative performance of different methods. For Hamming ranking based evaluation, we compute the retrieval precision as the percentage of true neighbors among the top M returned samples, where M is uniformly set as 500 in the experiments. Finally, similar to [13], a Hamming radius of 2 is used to retrieve the neighbors in the case of hash lookup. The precision of the returned samples falling within Hamming radius 2 is reported. If a query returns no neighbors inside Hamming ball with radius 2, it is treated as a failed query with zero precision.

5.2 Datasets

We used three image datasets in our experiments, i.e., CIFAR-10, a Flickr image collection, and the 80 million tiny images, with the number of samples ranging from tens of thousands to millions. In addition, we use 1 million SIFT feature vectors for the experiments with unsupervised hashing methods. For the first two datasets, since they are fully annotated, we focus on the quantitative evaluation of the search accuracy. For the 80 million image data, we demonstrate the scalability of the proposed methods and provide qualitative evaluation by providing the search results of some exemplar queries.

CIFAR-10 dataset

The CIFAR-10 dataset is a labeled subset of the 80-million tiny images collection [32]. It consists of a total of 60000 32×32 color images in 10 classes, each of which has 6000 samples.[1] Each sample in this dataset is associated with a mutually exclusive class label. A few example images from CIFAR10 dataset are shown in Figure 3. Since this dataset is fully annotated, the ground truth semantic neighbors can be easily retrieved based on the class labels. The entire dataset is partitioned into two parts: a training set with 59000 samples and a test set with 1000 samples. The training set is used for learning hash functions and constructing the hash look-up tables. For *BSSC*, *BRE* and the proposed semi-supervised hashing methods, we additionally sample 1000 random points from the training set and assign semantic nearest neighbor information (i.e. construct the pairwise label matrix $\mathbf{S}$) based on the image labels. The binary encoding is performed in the 384-dim GIST feature space [33].

1. http://www.cs.toronto.edu/ kriz/cifar.html

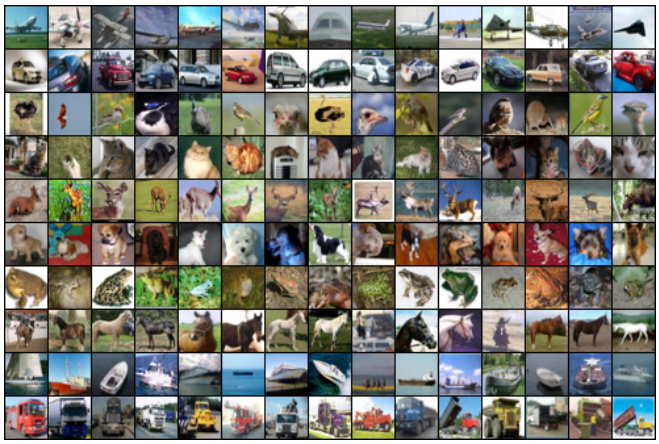


Fig. 3. A few example images from the CIFAR10 dataset. From top row to bottom row, the image classes are *airplane*, *automobile*, *bird*, *cat*, *deer*, *dog*, *frog*, *horse*, *ship*, and *truck*.

Flickr Images

Here we use a set of Flickr consumer images collected by NUS lab, i.e. NUS-WIDE dataset [34]. It was created as a benchmark for evaluating multimedia search techniques. This dataset contains around 270000 images associated with 81 ground truth concept tags. Unlike the CIFAR10 dataset, each sample in NUS-WIDE could be assigned multiple labels, since this kind of multiple tagging occurs very often in real-world annotation scenario. Compared to the CIFAR-10 dataset, NUS-WIDE dataset contains images with much higher resolutions, which allow us to use a Bag-of-Visual-Word model with local SIFT features for extracting the image descriptor [35]. Particularly, a visual vocabulary with 500-length code book and a soft assignment strategy was used for deriving the image features, as described in [36].

Similar to CIFAR10 dataset, we partitioned this set into two parts, 1K for query test and around 269K for training and constructing hash tables. In addition, $2K$ images are randomly sampled with labels for *BSSC*, *BRE* and our semi-supervised hashing methods. The precision is evaluated based on whether the returned images and the query share *at least one* common semantic label. The performance was evaluated with different code lengths varying from 8-bit to 64-bit.

SIFT-1M Dataset

We also test the performance of our unsupervised sequential learning method (*USPLH*) using SIFT-1M dataset. It contains 1 million local SIFT descriptors extracted from a large set of images described in [28]. Each point in the dataset is a 128-dim vector representing histograms of gradient orientations. We use 1 million samples for training and additional $10K$ for testing. Euclidean distance is used to determine the nearest neighbors. Following the criterion used in [37][13], a returned point is considered a good neighbor if it lies in the top 2 percentile points closest to a query. Since no labels are available in this experiment, both SSH_{orth} and $SSH_{nonorth}$ have no adjustment term. Because it results in the same hash functions by using just principal projections, we named it as PCA based hashing (*PCAH*). We also compared with a few unsupervised hashing techniques, including *LSH*, *SH*, *SIKH* on this dataset. For *USPLH*, to learn each hash function sequentially, we select 2000 samples from each of the four boundary and margin regions r^-, r^+, R^-, R^+. A label matrix $\mathbf{S}$ is constructed by assigning pseudo-labels to pairs generated from these samples.

80 Million Tiny Images

Besides the quantitative evaluation on the above three datasets, we also apply our techniques on a large collection of images with Gist features, i.e., 80 million tiny images dataset [32], which has been used as a benchmark dataset for designing binary encoding approaches [13][21][38][23]. However, only a small portion of the dataset is manually labeled and the associated meta information is fairly noisy. Since CIFAR10 is a fully annotated subset of this gigantic image collection, we combine these two datasets in our experiments. The experimental protocol for evaluation is described as below. A subset of two million data points is sampled to construct the training set, especially for computing the data covariance matrix for all the eigen-decomposition based approaches, and a separate set of 2K samples from CIFAR10 dataset is used as labeled samples. For *SH*, the hash functions were designed using this two-million dataset. For *BRE* and the proposed semi-supervised hashing methods, both the two-million dataset and 2K labeled data were used to learn the hash functions. After obtaining the hash functions, the Hamming embedding of the *entire* dataset with a total of $79,302,017$ samples is computed with 64-bit hash codes. Finally, some examples are randomly selected for query test, and qualitative comparison is made with other methods.

5.3 Results

For quantitative evaluation on CIFAR10 and Flickr datasets, the number of bits is varied from 8 to 48 for CIFAR10 dataset and Flickr image dataset. The performance curves are shown in Figure 4 and 5, respectively, where the precision curves of both Hamming ranking and hash lookup (within Hamming radius 2) show accuracy of the returned top results. From these figures, it is not very surprising to see that *LSH* provides the worst performance since the random hash functions lack discrimination for small bit lengths. The orthogonal solution for *SSH* described in Section 4, i.e. SSH_{orth}, has comparable performance to the non-orthogonal solution, $SSH_{nonorth}$, for small number of bits (i.e. 8, 12, and 16 bits) since there is enough variance in top few orthogonal directions computed in our semi-supervised formulation. But when using large number of bits, SSH_{orth} performs much worse since the orthogonal solution forces one to progressively pick the low variance projections, substantially reducing the quality of the whole embedding. Both Figure 4 and 5 clearly show that $SSH_{nonorth}$ is significantly better than SSH_{orth} when using long hash codes. Note that although *SH* also uses principal projections directions, it can somewhat alleviate the negative impact of low variance projections by reusing the large variance projections with higher frequency sinusoidal binarization. The sequential method of *SSH*, i.e., *SPLH*, provides the best performance for

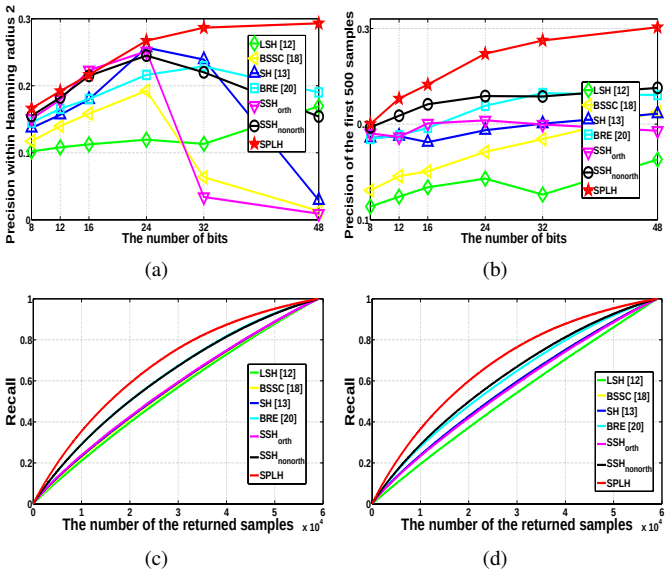


Fig. 4. Results on CIFAR10 dataset. a) Precision within Hamming radius 2 using hash lookup; b) Precision of the top 500 returned samples using Hamming ranking; c) Recall curves with 24 bits; d) Recall curves with 32 bits. *LSH*: Locality Sensitive Hashing, *BSSC*: Boosted Similarity Sensitive Coding, *SH*: Spectral Hashing, *BRE*: Binary Reconstructive Embedding, SSH_{orth}: Orthogonal Semi-Supervised Hashing, $SSH_{nonorth}$: Non-orthogonal Semi-Supervised Hashing, and *SPLH*: Sequential Projection Learning based Hashing.

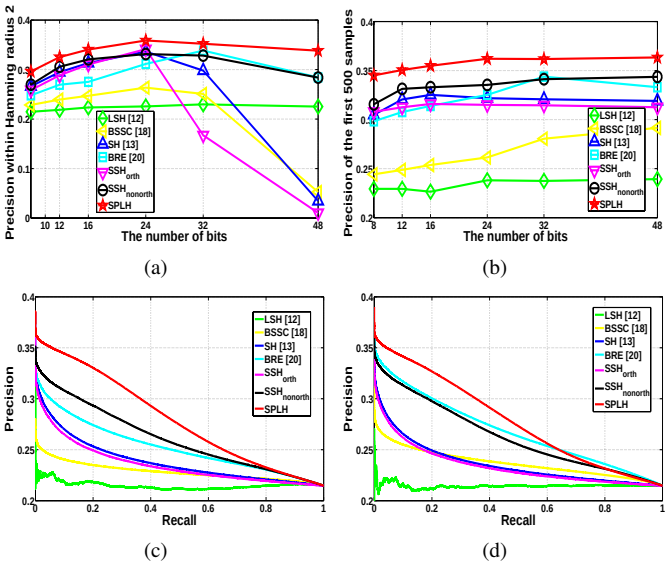


Fig. 5. Results on Flickr image dataset. a) Precision within Hamming radius 2 using hash lookup; b) Precision of the top 500 returned samples using Hamming ranking; c) Precision-Recall curve with 24 bits; d) Precision-Recall curve with 32 bits. *LSH*: Locality Sensitive Hashing, *BSSC*: Boosted Similarity Sensitive Coding, *SH*: Spectral Hashing, *BRE*: Binary Reconstructive Embedding, SSH_{orth}: Orthogonal Semi-Supervised Hashing, $SSH_{nonorth}$: Non-orthogonal Semi-Supervised Hashing, and *SPLH*: Sequential Projection Learning based Hashing.

all bits. Particularly, in the evaluation of hash lookup within Hamming radius 2 (Figure 4(a) and 5(a)), the precision for most of the compared methods drops significantly when longer codes are used. This is because, for longer codes, the number of points falling in a bucket decrease exponentially. Thus, many queries fail by not returning any neighbor even in a Hamming ball of radius 2. This shows a practical problem with hash lookup tables even though they have faster query response than Hamming ranking. Even in this case, *SPLH* provides the best performance for most of the cases. Also, the drop in precision for longer codes is much less compared to others, indicating less failed queries for *SPLH*.

As a complementary evaluation, the recall curves for CIFAR10 set are given in Figure 4(c) and 4(d), and the precision-recall curves for Flickr set are given in Figure 5(c) and 5(d). The results demonstrate significant performance improvement using the proposed semi-supervised hashing approaches, especially *SPLH*, over other methods. Different from the CIFAR10 dataset, the Flickr dataset contains images with multiple labels. Therefore, the performance measure solely based on recall is incomplete for this multi-label problem [39]. Instead, average precision, approximated by the area under the precision-recall curves, is a more appropriate measure.

We implemented the proposed methods and other hashing approaches in Matlab and ran the experiments on a Lenovo workstation with 3.16 GHz Quad Core CPU. Figure 6 reports the comparison of the computational cost, including training time and compression time, for different techniques. The training time indicates the cost of learning the hash functions from training data and the compression time measures the encoding time from the original test data to binary codes. It is not surprising that *LSH* needs negligible training time since the projections are randomly generated, instead of being learned. The three eigenvalue decomposition based techniques, i.e. *SH*, SSH_{orth}, and $SH_{nonorth}$, incur similar training cost. Since *SPLH* needs to update pairwise label matrix and performs eigenvalue decomposition at each iteration, its training time is longer but comparable to *BRE*, and much less than *BSSC*. Compared with off line training cost, the compression time is usually more important in practice since it is done in real time. As shown in Figure 7(b) and 6(d), *BRE* is the most expensive method in terms of computing the binary codes. *SH* requires a little more time than the remaining methods due to the calculation of the sinusoidal function. The code generation time can be ranked as: $BRE \gg SH > LSH \simeq BSSC \simeq SSH_{orth} \simeq SSH_{nonorth} \simeq SPLH$.

In all the above experiments, we fixed the size of the training subset, e.g., 2000 for the Flickr dataset. To study the impact of the amount of supervision on the search performance, we conducted further experiments on the Flickr dataset using 24-bit hash functions while varying the number of training samples from 1000 to 3000. Since *BRE* achieved comparable performance, we also compared against its performance in these tests. Figure 7 gives the precision curves for both hash lookup and Hamming ranking for the proposed semi-

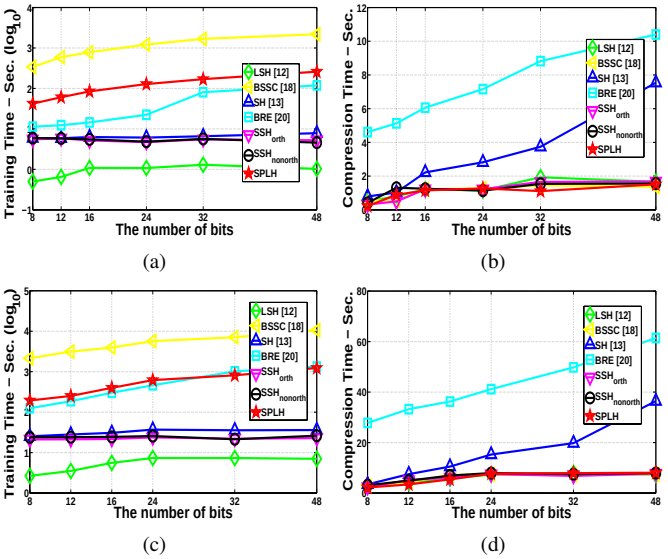


Fig. 6. Computational cost for different binary encoding methods. a) Training cost on CIFAR10 dataset; b) Compression cost on CIFAR10 dataset; c) Training cost on Flickr dataset; d) Compression cost on Flickr dataset. *LSH*: Locality Sensitive Hashing, *BSSC*: Boosted Similarity Sensitive Coding, *SH*: Spectral Hashing, *BRE*: Binary Reconstructive Embedding, SSH_{orth}: Orthogonal Semi-Supervised Hashing, $SSH_{nonorth}$: Non-orthogonal Semi-Supervised Hashing, and *SPLH*: Sequential Projection Learning based Hashing.

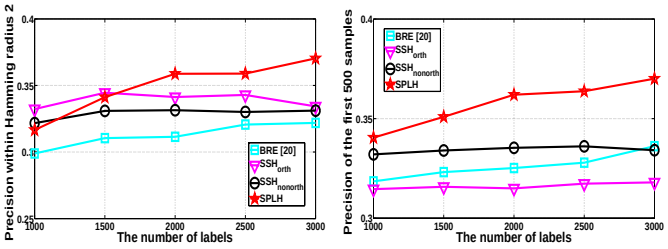


Fig. 7. Evaluation of the performance using different amounts of training samples. a) Precision within Hamming radius 2 using hash lookup on Flickr dataset; b) Precision of the top 500 returned samples using Hamming ranking on Flickr dataset. *BRE*: Binary Reconstructive Embedding, SSH_{orth}: Orthogonal Semi-Supervised Hashing, $SSH_{nonorth}$: Non-orthogonal Semi-Supervised Hashing, and *SPLH*: Sequential Projection Learning based Hashing.

supervised methods and supervised *BRE* approach. It is clear that 2000 points were sufficient for the Flickr dataset to obtain reasonable performance for most of the approaches. Further adding more training data increases the training cost without adding much benefit.

Figure 8 shows the experimental results of the unsupervised tests on SIFT-1M dataset. Figure 8(a) shows precision curves for different methods using hash lookup table, and Figure 8(b) shows the precision curves using Hamming ranking. Methods that learn data-dependent projections i.e., *USPLH*, *SH* and *PCAH*, perform generally much better than *LSH* and *SIKH*. *SH* performs better than *PCAH* for longer codes since, for this dataset, *SH* tends to pick the high-variance directions again. *USPLH* gives the best performance for most cases. Also, for Hamming radius lookup experiments, the performance of *USPLH* does not drop as rapidly as SH and *PCAH* with increase in bits. Thus, *USPLH* leads to less query failures in comparison to other methods. Figure 8(c) and 8(d) show the recall curves for different methods using 24-bit and 48-bit codes. Higher precision and recall for *USPLH* indicate the advantage of learning hash functions sequentially even with noisy pseudo labels.

Finally, we present the experimental results on the large 80-million image data set to show the scalability of the proposed semi-supervised hashing methods. We use 64-bit codes to index the 384-dim Gist descriptor, which dramatically reduces the storage of the entire dataset from hundreds of gigabytes to a few hundred megabytes. A random set of queries was sampled from the database and used for tests. Here we compared the visual search results of the three best performing methods, i.e., *BRE*, $SSH_{nonorth}$, and *SPLH*. After obtaining the search results in hash lookup (within Hamming radius $r = 2$), we computed the Euclidian distance of the collected nearest neighbors and query images in Gist feature space and then sorted the results. The top ten returned images for a few exemplar queries are shown in Figure 9. *SPLH* presents more visually consistent search results than *BRE* and SSH_{orth}. This exhaustive search usually involved only around $0.005\% \sim 0.01\%$ samples from the entire 80 million set leading to dramatic reduction in search time."

6 Conclusions and Future Work

In this paper, we have proposed a semi-supervised paradigm to learn efficient hash codes by simple linear mapping which can handle semantic similarity/dissimilarity among the data. The proposed method minimizes empirical loss over the labeled data coupled with an information theoretic regularizer over both labeled and unlabeled data. A series of relaxations lead to a very simple eigen-decomposition based solution which is extremely efficient. Based on this framework, we proposed the following family of solutions:

1. **Orthogonal hash functions** (SSH_{orth}): By adding orthogonality as hard constraints, the hash codes can be directly obtained by conducting eigen-decomposition over an adjusted covariance matrix.
2. **Non-orthogonal hash functions** ($SSH_{nonorth}$): The orthogonality constraints can be relaxed as a soft penalty term in the objective function. Then, an *approximate* non-orthogonal solution can be obtained through adjusting the learned orthogonal solution.
3. **Sequential hash functions** (*SPLH*): The sequential method iteratively learns new hash functions such that in each iteration new function tends to minimize the errors made by the previous one. For this, the pairwise labels

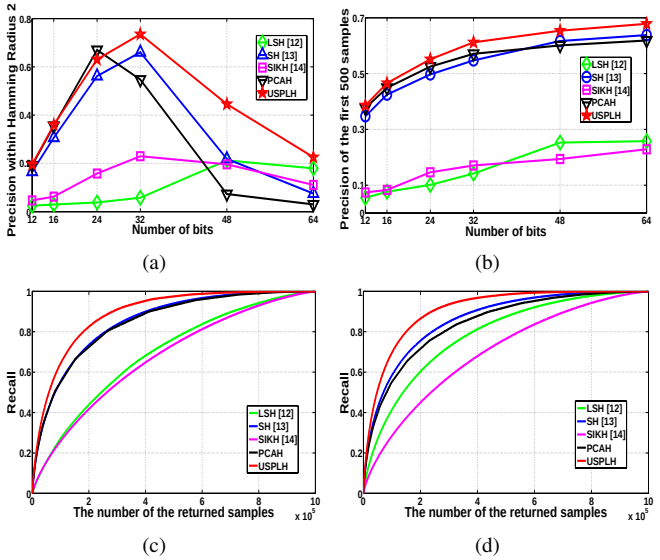


Fig. 8. Results on SIFT-1M dataset. a) precision of the top 500 returned samples using Hamming ranking; b) precision within Hamming radius 2 using hash lookup. Recall curves (c) with 24 bits, and (d) with 48 bits. *LSH*: Locality Sensitive Hashing, *SH*: Spectral Hashing, *SIKH*: Shift Invariant Kernel based Hashing, and *USPLH*: Unsupervised Sequential Projection Learning based Hashing.

are updated by imposing higher weights on point pairs violated by the previous hash function.

4. **Unsupervised sequential hash functions** (*USPLH*): The sequential learning method was extended to the unsupervised setting, where a set of pseudo-labels are generated sequentially using the probable mistakes made by the previous bit. Each new hash function tries to minimize these errors subject to the same regularizer as in the semi-supervised case.

We conducted extensive experiments on four large datasets containing up to 80 million points and provided both quantitative and qualitative comparison with the state-of-the-art hashing techniques. The experimental results show superior performance of the proposed semi-supervised hashing methods. Particularly, *SPLH* achieved the best performance for semi-supervised and supervised cases and *USPLH* performs the best for unsupervised cases. The sequential techniques, i.e. *SPLH* and *USPLH*, need more time for (offline) training than *LSH* or eigen-decomposition based methods such as *SH*, SSH_{orth}, and $SSH_{nonorth}$, but comparable to or even faster than *BRE* method, and much faster than *BSSC*. In terms of (online) run time, all the four proposed hashing methods are as fast as *LSH*, which is significantly faster than *SH* and *BRE* methods. In the future, we would like to investigate if any theoretical guarantees could be provided for the proposed semi-supervised hashing methods.

ACKNOWLEDGMENTS

We thank the reviewers for their helpful comments and insights. J. Wang was supported in part by Google Intern Scholarship and a Chinese Government Scholarship for Outstanding Self-Financed Students Abroad. S.-F. Chang was supported in part by National Science Foundation Awards CNS-07-51078 and CNS-07-16203.

REFERENCES

[1] R. Datta, D. Joshi, J. Li, and J. Z. Wang, "Image retrieval: Ideas, influences, and trends of the new age," *ACM Computing Surveys*, vol. 40, no. 2, pp. 1–60, 2008.

[2] G. Shakhnarovich, T. Darrell, and P. Indyk, *Nearest-neighbor methods in learning and vision: theory and practice*. MIT Press, 2006.

[3] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proc. of 30th ACM Symposium on Theory of Computing*, 1998, pp. 604–613.

[4] J. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, p. 517, 1975.

[5] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, 1977.

[6] C. Silpa-Anan and R. Hartley, "Optimised kd-trees for fast image descriptor matching," in *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[7] S. Omohundro, "Efficient algorithms with neural network behavior," *Complex Systems*, vol. 1, no. 2, pp. 273–347, 1987.

[8] J. Uhlmann, "Satisfying general proximity/similarity queries with metric trees," *Information Processing Letters*, vol. 40, no. 4, pp. 175–179, 1991.

[9] P. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in *Proc. of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, 1993, pp. 311–321.

[10] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Applications*, Algarve, Portugal, 2009, pp. 331–340.

[11] P. Indyk, "Nearest-neighbor searching in high dimensions," in *Handbook of discrete and computational geometry*, J. E. Goodman and J. O'Rourke, Eds. Boca Raton, FL: CRC Press LLC, 2004.

[12] A. Gionis, P. Indyk, and R. Motwani, "Similarity search in high dimensions via hashing," in *Proc. of 25th International Conference on Very Large Data Bases*, 1999, pp. 518–529.

[13] Y. Weiss, A. Torralba, and R. Fergus, "Spectral hashing," in *Proc. of Advances in Neural Information Processing Systems*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., 2008, vol. 21, pp. 1753–1760.

[14] M. Raginsky and S. Lazebnik, "Locality-sensitive binary codes from shift-invariant kernels," in *Advances in Neural Information Processing Systems 22*, Y. Bengio, D. Schuurmans, J. Lafferty, C. K. I. Williams, and A. Culotta, Eds., 2009, pp. 1509–1517.

[15] B. Kulis and K. Grauman, "Kernelized locality-sensitive hashing for scalable image search," kyoto, Japan, 2009, pp. 2130 – 2137.

[16] W. Liu, J. Wang, S. Kumar, and S.-F. Chang, "Hashing with graphs," in *Proceedings of the 28th International Conference on Machine Learning*, 2011, pp. 1–8.

[17] A. Smeulders, M. Worring, S. Santini, A. Gupta, and R. Jain, "Content-based image retrieval at the end of the early years," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 22, no. 12, pp. 1349–1380, 2000.

[18] B. Kulis, P. Jain, and K. Grauman, "Fast similarity search for learned metrics," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 12, pp. 2143–2157, 2009.

[19] G. Shakhnarovich, "Learning task-specific similarity," Ph.D. dissertation, Massachusetts Institute of Technology, 2005.

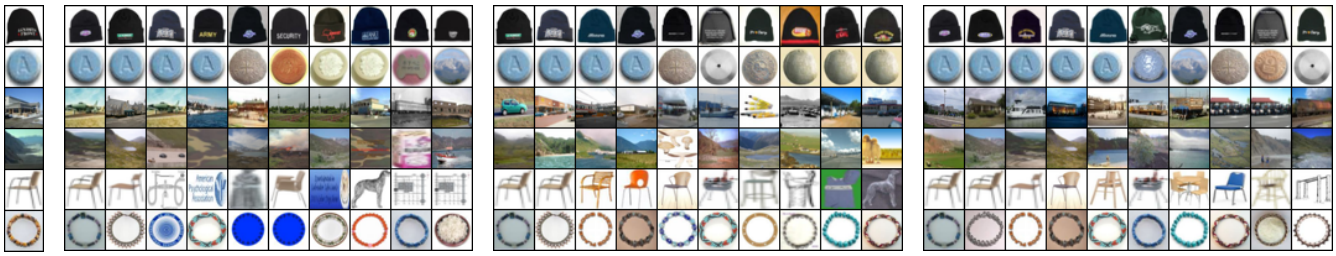
[20] G. Hinton and R. Salakhutdinov, "Reducing the dimensionality of data with neural networks," *Science*, vol. 313, no. 5786, pp. 504–507, 2006.

[21] B. Kulis and T. Darrell, "Learning to Hash with Binary Reconstructive Embeddings," in *Proc. of Advances in Neural Information Processing Systems*, vol. 20.

[22] Y. Mu, J. Shen, and S. Yan, "Weakly-Supervised Hashing in Kernel Space," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Francisco, USA, June, pp. 3344–3351.

[23] A. Torralba, R. Fergus, and Y. Weiss, "Small codes and large image databases for recognition," in *IEEE Conference on Computer Vision and Pattern Recognition*, Anchorage, Alaska, USA, 2008, pp. 1–8.

[24] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the twentieth annual Symposium on Computational Geometry*, 2004, pp. 253–262.



(a) (b) (c) (d)

Fig. 9. Qualitative evaluation over the 80 million tiny image dataset using 64-bit codes. a) query images; top 10 returned images using b) *BRE* method; c) $SSH_{nonorth}$ method, and d) *SPLH* method. *BRE*: Binary Reconstructive Embedding, $SSH_{nonorth}$: Non-orthogonal Semi-Supervised Hashing, and *SPLH*: Sequential Projection Learning based Hashing.

[25] M. Bawa, T. Condie, and P. Ganesan, "LSH forest: self-tuning indexes for similarity search," in *Proceedings of the 14th international conference on World Wide Web*, Chiba, Japan, 2005, pp. 651–660.

[26] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd international conference on Very large data bases*, 2007, pp. 950–961.

[27] L. Cayton and S. Dasgupta, "A learning framework for nearest neighbor search," in *Advances in Neural Information Processing Systems 20*.

[28] J. Wang, S. Kumar, and S.-F. Chang, "Sequential projection learning for hashing with compact codes," in *Proceedings of the 27th International Conference on Machine Learning*, 2010, pp. 1127–1134.

[29] Y. Freund and R. Schapire, "A desicion-theoretic generalization of on-line learning and an application to boosting," in *Computational Learning Theory*, 1995, pp. 23–37.

[30] C. Fowlkes, S. Belongie, F. Chung, and J. Malik, "Spectral grouping using the Nyström method," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 2, pp. 214–225, 2004.

[31] S. Baluja and M. Covell, "Learning to hash: forgiving hash functions and applications," *Data Mining and Knowledge Discovery*, vol. 17, no. 3, pp. 402–430, 2008.

[32] A. Torralba, R. Fergus, and W. Freeman, "80 million tiny images: A large data set for nonparametric object and scene recognition," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 30, no. 11, pp. 1958–1970, 2008.

[33] A. Oliva and A. Torralba, "Modeling the shape of the scene: A holistic representation of the spatial envelope," *International Journal of Computer Vision*, vol. 42, no. 3, pp. 145–175, 2001.

[34] 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*, Santorini, Greece, July 2009.

[35] D. G. Lowe, "Distinctive Image Features from Scale-Invariant Keypoints," *International Journal of Computer Vision*, vol. 60, no. 2, pp. 91–110, 2004.

[36] Y.-G. Jiang, C.-W. Ngo, and J. Yang, "Towards optimal bag-of-features for object categorization and semantic video retrieval," in *Proceedings of the 6th ACM international conference on Image and video retrieval*, 2007, pp. 494–501.

[37] J. Wang, S. Kumar, and S.-F. Chang, "Semi-supervised hashing for scalable image retrieval," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, San Francisco, USA, June 2010, pp. 3424–3431.

[38] R. Fergus, Y. Weiss, and A. Torralba, "Semi-supervised learning in gigantic image collections," in *Advances in Neural Information Processing Systems 22*.

[39] R. Schapire and Y. Singer, "Boostexter: A boosting-based system for text categorization," *Machine learning*, vol. 39, no. 2, pp. 135–168, 2000.

Jun Wang received the M.Phil. and Ph.D. degrees from Columbia University, NY, in 2010 and 2011, respectively. Currently, he is a Research Staff Member in the business analytics and mathematical sciences department at IBM T. J. Watson Research Center, Yorktown Heights, NY. He also worked as an intern at Google Research in 2009, and as a research assistant at Harvard Medical School, Harvard University in 2006. He has been the recipient of several awards and scholarships, including the Jury thesis award from the Department of Electrical Engineering at Columbia University in 2011, the Google global intern scholarship in 2009, and a Chinese government scholarship for outstanding self-financed students abroad in 2009. His research interests include machine learning, business analytics, information retrieval and hybrid neural-computer vision systems.

Sanjiv Kumar received his B.E. from Birla Institute of Science and Technology, Pilani, India and M.S. from Indian Institute of Technology, Chennai, India in 1997. From 1997 to 1999, he was a Research Fellow at the Department of Surgery, National University of Singapore working in the field of medical robotics and imaging. In 2000, he joined the Ph.D. program at The Robotics Institute, Carnegie Mellon University. Since 2005, he has been working at Google Research, NY as a Research Scientist. His primary research interests include large scale computer vision and machine learning, graphical models and medical imaging.

Shih-Fu Chang (S'89-M'90-SM'01-F'04) is Richard Dicker Professor in the Departments of Electrical Engineering and Computer Science, and Director of Digital Video and Multimedia Lab at Columbia University. He has made significant contributions to multimedia search, visual communication, media forensics, and international standards. He has been recognized with ACM SIGMM Technical Achievement Award, IEEE Kiyo Tomiyasu Award, Navy ONR Young Investigator Award, IBM Faculty Award, ACM Recognition of Service Award, and NSF CAREER Award. He and his students have received many Best Paper Awards, including the Most Cited Paper of the Decade Award from Journal of Visual Communication and Image Representation. He has worked in different advising/consulting capacities for industry research labs and international institutions. He is an IEEE Fellow and a Fellow of the American Association for the Advancement of Science. He served as Editor-in-Chief for IEEE Signal Processing Magazine (2006-8), and Chair of Columbia's Electrical Engineering Department (2007-2010).