Supervised Binary Hash Code Learning With Jensen Shannon Divergence

Lixin Fan
Nokia Research Center
fanlixin@ieee.org

Abstract

This paper proposes to learn binary hash codes within a statistical learning framework, in which an upper bound of the probability of Bayes decision errors is derived for different forms of hash functions and a rigorous proof of the convergence of the upper bound is presented. Consequently, minimizing such an upper bound leads to consistent performance improvements of existing hash code learning algorithms, regardless of whether original algorithms are unsupervised or supervised. This paper also illustrates a fast hash coding method that exploits simple binary tests to achieve orders of magnitude improvement in coding speed as compared to projection based methods.

1. Introduction

Mapping high dimensional image data into binary codes plays an indispensable role for efficient storing and searching of large scale image databases. Owing to storage efficiency and sublinear search time, compact binary codes have been widely applied to various vision applications such as object recognition [20], image retrieval [23], local descriptor matching [6, 19] and binary feature matching [2, 9, 17, 21] etc.

Learning compact binary code has been traditionally treated as a "similarity-preserving" problem with objective to *map similar data points into similar binary codes*. Often original data points are assumed to reside in a high-dimensional Euclidean space while (dis-)similarity between binary codes is quantified by Hamming distance. The learning process then aims to minimize discrepancies between pairwise Euclidean and Hamming distances. This approach is well suited for approximate nearest neighbour search in Euclidean space e.g. as in [26, 7].

Recent work demonstrate considerable performance improvements with a discernible shift of research focus to (semi-)supervised approaches, in which the learning of hash functions is invariably governed by some forms of label information. By exploiting either semantic labels provided by human or labels derived from neighbourhoods in Euclidean space, these supervised approaches optimize a variety of objective functions including empirical accuracy of (dis-)similar data points [24, 23, 12], pairwise hinge-like loss [13], triplet ranking loss [14] and binary code quantization loss [3] etc. Despite empirical justifications with large databases, the choice of objective functions for these methods is to some extent heuristic and it remains an open question to exploit label information in a consistent and theoretically sound manner.

The research presented in this paper, therefore, proposes to formulate binary hash code learning within a statistical learning framework, in which *an upper bound of the probability of Bayes decision errors* is derived for different forms of hash functions (Section 3). Furthermore, a rigours proof of the convergence of the upper bound for arbitrary *sequential* learning algorithms is presented in Theorem 1. Consequently, minimizing the upper bound for existing hash code learning algorithms leads to consistent performance improvements, regardless of whether original algorithms are unsupervised or supervised (Section 4).

The processing time of converting a query data point into binary codes (i.e. *coding time*) is a crucial performance parameter for nearest neighbour search algorithms. Having short coding time is of particular interest for vision applications such as fast keypoint recognition [16] and image localization [18] etc, in which query images may contain up to 10K features and the coding time amounts to a significant portion of the total processing time. While the coding time is considered a constant for projection based hash functions, we demonstrate in Section 4.4 that it actually can be reduced by order of magnitudes by exploiting simple yet discriminative *binary tests* of input feature vectors. The proposed Haar-Like hash function, albeit suboptimal in terms of precision rate, is preferred due to its high speed and computational simplicity for real-time applications running on mobile devices.

2. Related Work

We review below only supervised hash code learning methods and refer readers to two journal articles [19, 4] for thorough reviews of this active research area.

Supervised hash learning methods: Kullis and Darrell proposed to minimize reconstruction errors between both *similar* and *dissimilar* data points [7]. Wang et al. [23, 24] and Liu et al. [12] used a supervised term to minimize empirical errors between data point pairs associated with three types of label i.e. *similar*, *dissimilar* and *others*. Motivated by the hinger loss in SVMs, Norouzi and Fleet proposed to minimize the upper bound on empirical loss of similar (or dissimilar) point pairs [13], and extended the method to incorporate *ranking loss* defined on triplets of binary codes [14]. Similar loss function was used in [10] to encode proximity comparison information of labelled data. Recently, Wang et al. generalized the idea to *preserve orders* of multiple sub-categories defined on training data, and demonstrated favourable results in comparison with state of the art methods [25].

Binary tests: There are abundant literatures related to *binary test* descriptors. Owing to its fast calculation speed, binary test has long been adopted to extract Haar-like features for real-time object detection [22]. Simple binary tests were used for fast keypoint recognition in a Naive Bayesian classification framework [16], which is probably the most similar work to the proposed Haar-Like hash functions, nevertheless, the random selection of binary tests adopted in [16] often led to low precision rates for short code lengths (see Section 4.4 of this paper for a comparison). BRIEF descriptor completely abandoned the training phase, but had to resort to long code lengths ranging from 128 to 256 bits [2]. More recent ORB descriptor decorrelated BRIEF features by using a learning step to search for binary tests with means near 0.5 [17]. D-Brief descriptors [21] was trained to reduce code lengths while maintaining high discriminative power.

Jensen Shannon Divergence (JSD) used in this paper is related to information gain (IG), which was used to select the "best" random test when growing a randomized forest classifier [8, 1]. However, IG-based selection was found not necessarily better than random selection in those papers.

3. Theoretic Analysis

This section lays down theoretic foundation for a sequential binary hash code learning framework and presents a rigorous proof of the convergence property of the framework.

3.1. A Statistical Learning Framework for Binary Code Learning

We treat learning optimal binary codes as a typical *multi-class classification* problem, in which a p-dimensional observation $\mathsf{x} = (x_1, x_2, ..., x_p) \in \mathbb{R}^p$ is classified as coming from one of M possible classes $C_1, C_2, ..., C_M$. Let $\pi_1, ..., \pi_M$ denote the a priori probabilities of the classes and $p_1(\mathsf{x}), ..., p_M(\mathsf{x})$ denote corresponding probability density functions. For now let us assume these are known. Nevertheless, we do NOT assume any specific forms of class distribution models e.g. Gaussian or uniform.

A hash function $h : \mathbb{R}^p \rightarrow \{0, 1\}$ is often treated as a mapping of an observation x to a single bit binary code. Depending on the outcome of the hash function, the entire sample space $S = \mathbb{R}^p$ is partitioned into two complementary B-subsets that are defined as follows:

$$[b]_h^S = \{\mathsf{x} \in S \mid h(\mathsf{x}) = b\}, b = 0 \text{ or } 1. \tag{1}$$

By definition, $[0]_h^S \cup [1]_h^S = S$, $[0]_h^S \cap [1]_h^S = \emptyset$ and specially $[\emptyset]_\emptyset^S = S$. This definition of B-subsets is generic and accommodates to different families of hash functions such as linear transform, kernelized or more complex hash functions used e.g. in [5, 26, 12].

A set of K hash functions $\mathbb{h}_K = \{h_1, h_2, ..., h_K\}$ partitions the space S into 2^K non-overlapping B-subsets[1], which are intersections of B-subsets of each hash function:

$$[b_1 b_2 ... b_K]_{\mathbb{h}_K}^S = [b_1]_{h_1}^S \cap [b_2]_{h_2}^S \cap ... [b_K]_{h_K}^S.$$

Each B-subset $[b_1 b_2 ... b_K]_{\mathbb{h}_K}^S$ is uniquely determined by its binary code $[b_1 b_2 ... b_K]$ and partitioning hash functions $\mathbb{h}_K$. When ambiguity seems unlikely, $\mathbb{h}_K$ and S are omitted and binary codes are denoted as $b_{1...K}$ for brevity of description.

If a hash code $[b_1 b_2 ... b_K]$ is observed, the posterior probability of class C_m is given by Bayes Theorem:

$$\Pr(C_m | b_{1...K}) = \frac{\pi_m \int_{b_{1...K}} p_m(\mathsf{x}) d\mathsf{x}}{\sum_{i=1}^{M} \pi_i \int_{b_{1...K}} p_i(\mathsf{x}) d\mathsf{x}}.$$

The probability of Bayes decision error by choosing the class MAP estimate $C_{\tilde{m}}$ that has the largest posterior probability is:

$$P(e \mid b_{1...K}) = 1 - \frac{\pi_{\tilde{m}} \int_{b_{1...K}} p_{\tilde{m}}(\mathsf{x}) d\mathsf{x}}{\sum_{i=1}^{M} \pi_i \int_{b_{1...K}} p_i(\mathsf{x}) d\mathsf{x}},$$

and the total probability of error for a given set of hash functions $\mathbb{h}_K$ is:

$$P(e \mid \mathbb{h}_K) = \sum_{b_{1...K} \in D_{\mathbb{h}_K}} P(e \mid b_{1...K}),$$

where $D_{\mathbb{h}_K}$ is a finite set with all non-empty B-subsets as its members.

Our goal is to seek a set of hash functions that minimizes the total probability of Bayes decision errors:

$$\mathbb{h}_K^* = \arg\min_{\mathbb{h}_K} P(e \mid \mathbb{h}_K). \tag{2}$$

[1] Depending on different hash functions, certain binary codes may correspond to empty B-subsets. Nevertheless these empty sets can be simply skipped from consideration.

Minimizing the objective function (2) is akin to minimizing empirical training error in recent supervised hashing learning methods [24, 23, 12]. However, the non-smooth $\max(.)$ function in the class MAP estimate makes it difficult to analyze the convergence property of the minimization of (2). In the rest of the paper, we first present a rigorous proof of the convergence of an upper bound on $P(e)$ and then illustrate how to use the upper bound to supervise a variety of hash code learning algorithms.

3.2. Jensen Shanon Divergence

It has been shown, by Theorem 6 in [11], that the probability of error $P(e)$ is related to the generalized Jensen-Shannon Divergence (JSD):

$$P(e) \leq \frac{1}{2}\Big(H(\pi) - \mathit{JSD}_{\pi}(p_1, ..., p_M)\Big)$$

where $H(\pi) = -\sum_{i=1}^{M} \pi_i \ln \pi_i$ is the entropy of the a priori probabilities and

$$\begin{aligned} \mathit{JSD}_{\pi}(p_1, ..., p_M) &= H\Big(\sum_{i=1}^{M} \pi_i p_i\Big) - \sum_{i=1}^{M} \pi_i H(p_i) \\ &= \sum_{i=1}^{M} \pi_i KL(p_i||\bar{p}). \end{aligned} \tag{3}$$

JSD can also be interpreted as weighted, by π_i, average of Kullback-Leibler divergences $KL(p_i||\bar{p})$ between class distributions and the *mixture distribution* $\bar{p} = \sum_{i=1}^{M} \pi_i p_i$.

For a given set of hash functions $\mathbb{h}_K$, JSD is in discrete form and can be computed by summing over all B-subsets:

$$\mathit{JSD}_{\pi}(p_1, ..., p_M \mid \mathbb{h}_K) = \sum_{b_{1...K} \in D_{\mathbb{h}_K}} \mathit{JSD}_{\pi}(p_1^{b_{1...K}}, ..., p_M^{b_{1...K}}), \tag{4}$$

where $p_i^{b_{1...K}} = \int_{b_{1...K}} p_i(\mathsf{x})d\mathsf{x}$. Again, we may abuse above notations as $\mathit{JSD}_{\pi}^{\mathbb{h}_K}$ and $\mathit{JSD}_{\pi}^{b_{1...K}}$ respectively.

Since $H(\pi)$ is a constant for a given problem, it immediately follows that the upper bound of the probability of error is minimized by maximizing (4):

$$\mathbb{h}_K^* = \arg\max_{\mathbb{h}_K} \mathit{JSD}_{\pi}(p_1, ..., p_M \mid \mathbb{h}_K). \tag{5}$$

Remark 1: The JSD measure as an objective function can be intuitively interpreted as the combination of an *unsupervised* and a *supervised* terms: maximizing the first term $H\Big(\sum_{i=1}^{M} \pi_i p_i\Big)$ in (3) leads to *balanced* bit assignments regardless of class labels. Maximizing the second term $-\sum_{i=1}^{M} \pi_i H(p_i)$, on the other hand, often leads to *unbalanced* bit assignments among different classes. These two opposing terms conspire to minimize the upper bound of the probability of MAP decision errors.

Algorithm 1: LSH-JSD Sequential Learning

Input: $\mathbf{x} \in \mathbb{R}^{N \times p}$, $y_n \in \{1, ..., M\}$, $n = 1, ..., N$, K, L
begin
  $\mathbb{h}_K = \emptyset$; $\pi_i = \frac{N_i}{N}$; $\mathbf{I}_i^{\emptyset} = \mathbf{1}^{N_i \times 1}$; where
  $N_i :=$ #data points in class i;
  $h_l = \text{sgn}(\mathbf{x}\mathbf{w}_l)$ and $\mathbf{H} = \text{sgn}(\mathbf{x}W)$, where
  $W = [\mathbf{w}_1, ..., \mathbf{w}_L] \in \mathbb{R}^{p \times L}$ are candidate projections;
  for $k = 1; k \leq K; k = k + 1$; **do**
    Compute $\mathit{JSD}_{\pi}^{\mathbb{h}_K \cup h_l}$ for all h_l by looping
    for *all* $b_{1...K}$ *with* $p_i^{b_{1...K}} \neq 0$ **do**
      $p_i^{b_{1...K} \cap [1]_{h_l}} \leftarrow$ Count "1" bits of $\mathbf{I}_i^{b_{1...K}} \cap h_l$;
    end
    Search h_l^* such that $\mathit{JSD}_{\pi}^{\mathbb{h}_K \cup h_l^*}$ is maximized;
    Update $\mathbb{h}_K \longleftarrow \mathbb{h}_K \cup h_l^*$;
    Update $\mathbf{I}_i^{b_{1...K}}$ accordingly;
  end
end
Output: $\mathbb{h}_K = \{h_1, .., h_K\}$

Remark 2: Owing to the convexity of KL divergence, Theorem 1 below proves that the upper bound of the Bayesian decision error is *monotonically decreasing* when a *sequential* learning approach repeatedly adds more hash functions except for some pathological cases.

Theorem 1. *Given a set of hash functions* $\mathbb{h}_K = \{h_1, .., h_K\}$ *and its superset* $\mathbb{h}_S = \{h_1, .., h_K, h_S\} \supset \mathbb{h}_K$, *if the condition in the proof below is satisfied, then*

$$\mathit{JSD}_{\pi}(p_1, ..., p_M \mid \mathbb{h}_K) < \mathit{JSD}_{\pi}(p_1, ..., p_M \mid \mathbb{h}_S).$$

Proof. $\forall\, [b_{1...K}]_{\mathbb{h}_K} = [b_{1...K}]_{\mathbb{h}_K} \cap \Big([0]_{h_S} \cup [1]_{h_S}\Big) = \Big([0]_{h_S} \cap [b_{1...K}]_{\mathbb{h}_K}\Big) \bigcup \Big([1]_{h_S} \cap [b_{1...K}]_{\mathbb{h}_K}\Big) \triangleq s_1 \bigcup s_2$.
Assume without loss of generality that $s_1 \neq \emptyset, s_2 \neq \emptyset$, we have $p_i^b \triangleq p_i^{b_{1...K}} = \int_{s_1} p_i(\mathsf{x})d\mathsf{x} + \int_{s_2} p_i(\mathsf{x})d\mathsf{x} \triangleq p_i^{s_1} + p_i^{s_2}$, $\overline{p^b} \triangleq \sum_{i=1}^{M} \pi_i p_i^b$, $\overline{p^{s_1}} \triangleq \sum_{i=1}^{M} \pi_i p_i^{s_1}$, $\overline{p^{s_2}} \triangleq \sum_{i=1}^{M} \pi_i p_i^{s_2}$.
It then follows the *log sum inequality* that

$$p_i^b \ln\Big(p_i^b / \overline{p^b}\Big) \leq p_i^{s_1} \ln\Big(p_i^{s_1} / \overline{p^{s_1}}\Big) + p_i^{s_2} \ln\Big(p_i^{s_2} / \overline{p^{s_2}}\Big) \tag{6}$$

with equality if and only if $p_i^{s_1} / \overline{p^{s_1}} = p_i^{s_2} / \overline{p^{s_2}}$.
Summing up LHS and RHS of (6) over all $[b_{1...K}]_{\mathbb{h}_K}$ and M classes, the theorem follows on the condition that: $\exists\, [b_{1...K}]_{\mathbb{h}_K} = s_1 \cup s_2$ such that $p_i^{s_1} / \overline{p^{s_1}} \neq p_i^{s_2} / \overline{p^{s_2}}$. □

Remark 3: the condition above is satisfied even for *random* hash functions and thus Theorem 1 also proves the convergence property of the well-known Locality Sensitive Hashing (LSH) method [5]. Nevertheless, as demonstrated

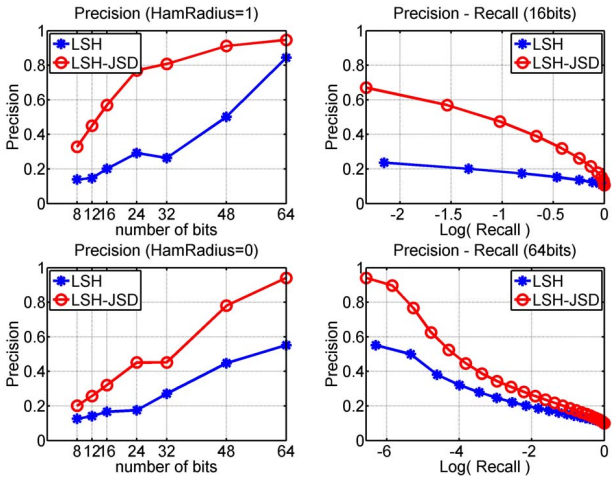

Figure 1. Comparison of LSH and LSH-JSD performances on CIFAR10 dataset. **Upper**: with Euclidean neighbours. **Bottom**: with Semantic labels. **Left**: Precision vs number of bits. **Right**: Precision vs Log(recall).

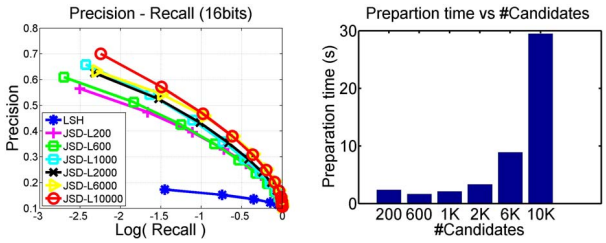

Figure 2. **Left**: Precision-Recall vs L (#candidate hash functions). **Right**: Preparation time vs L.

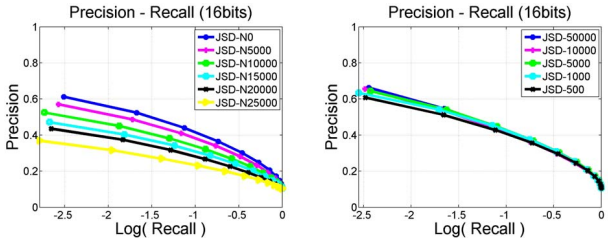

Figure 3. **Left**: Precision-Recall vs N (#random labels). **Right**: Precision-Recall vs #training labels.

in the following section, *to select the hash function which maximizes JSD at each step* is a preferred option to random selection of hash functions.

4. Experiments

Section 3 lays down theoretic foundation for a sequential binary hash code learning framework. In this section, we demonstrate how to adopt this generic framework to train existing learning algorithms with labelled datasets.

4.1. Dataset and Evaluation Protocols

We treat a dataset $\mathbf{x} \in \mathbb{R}^{N\times p}$ containing N p-dimensional row vectors $\mathbf{x}_n \in \mathbb{R}^{1\times p}, n = 1, ..., N$ as independent observations drawn from underlying multi-class distributions p_i, thus, class a priori probabilities can be directly estimated $\pi_i = \frac{N_i}{N}$ where N_i is the number of data points that belong to each class. For each data point, the associated class label $y_n \in \{1, ..., M\}$ is either a semantic label given by human or the one derived from Euclidean distance between data points. For the latter case, a *kmeans* clustering method is applied to obtain Euclidean neighbourhoods of the whole dataset.

Four publicly available datasets, namely *CIFAR10, CIFAR100, LabelMe22k* and *SIFT10K*, are used to compare the proposed hash code learning approach against existing methods. The *CIFAR10* dataset consists of 60K 32×32 colour images in 10 classes, with 5K training images and 1K test images, respectively, per class. Every images is represented by a 512-dimensional GIST feature vector [15]. Both semantic labels and Euclidean ones are used for *CIFAR10*. *CIFAR-100* is just like the *CIFAR-10*, except that it has 20 "coarse" and 100 "fine" superclasses. *LabelMe22K* contains 20K training images and 2K testing images. 50 classes and all 2000 testing images are used in our experiments. *SIFT10K* dataset contains 10K SIFT descriptors, which are classified into 100 classes. 100 nearest neighbours are provided for each class and there is one query SIFT descriptor per class that is used for testing in our experiments. In total, there are six sets of data/labels used in our experiments (see Table 4 for complete datasets).

Given a subset of testing data points with ground-truth labels, we use **Precision-Recall** curves to illustrate performances of different methods (see Figure 1 right column for examples) whereas *precision* and *recall* rates are defined as *the number of true positives* divided by, respectively, *the number of all retrieved* and *the number of all positives*. Two examples precision curves measured at Hamming radius 0 and 1 for hash codes lengths ranging from 8 to 64 bits are illustrated in Figure 1 (left column). Mean average precision (mAP) are often used to quantify performances of different methods. In this work, we measure mAP only for Hamming radius no greater than 3 since high precision rates prevent unnecessarily long lists of retrieved items.

4.2. Sequential Learning with JSD

We first demonstrate how to supervise the well-known LSH with a sequential learning algorithm 1. A set of L candidate linear projections $\mathbf{w}_l \in \mathbb{R}^{p\times 1}, l = 1, ...L$ are randomly generated and applied to the whole dataset $h_l = \text{sgn}(\mathbf{x}\mathbf{w}_l)$[2]. Outcomes of candidate projections are concatenated into a binary matrix $\mathbf{H} \in \{0, 1\}^{N\times L}$. We also rearrange $\mathbf{x}$ according to classes so that $\mathbf{H}$ can be partitioned into separated matrices $\mathbf{H}_i \in \{0, 1\}^{N_i\times L}$ per class.

[2]Following convention in literature, the bias term is absorbed into $\mathbf{w}_l$ for the brevity of description.

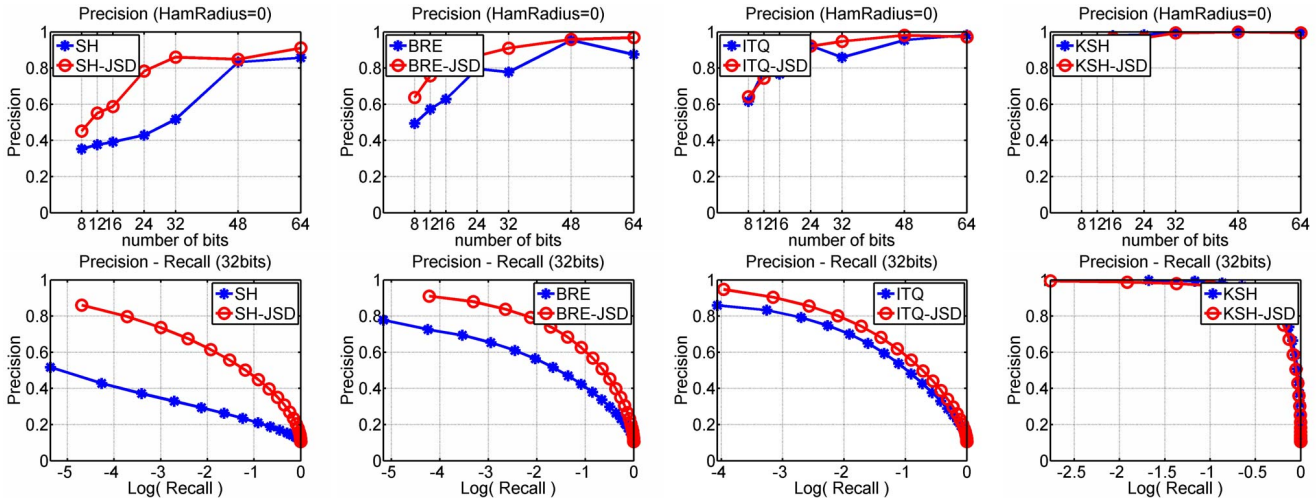

Figure 4. **Left to right**: Performance comparison of JSD with SH [26], BRE [7], ITQ [3] and KSH [12] on CIFAR10 dataset with Euclidean neighbourhood. **Upper**: Precision vs number of bits. **Bottom**: Precision vs Log(recall) with 32 bits code.

Binary vector $\mathbf{I}_i^{b_{1\dots K}} \in \{0,1\}^{N_i \times 1}$ indicates those points that are associated with class i and binary code $b_{1\dots K}$. This way $p_i^{b_{1\dots K} \cap [1]_{h_l}}$ can be efficiently computed by counting "1" bits in the intersection of $\mathbf{I}_i^{b_{1\dots K}}$ and corresponding column vectors h_l, and thereafter normalizing the count with respect to N_i. The complement $p_i^{b_{1\dots K} \cap [0]_{h_l}}$ can be computed accordingly. The whole JSD-based learning process is implemented as a binary tree growing algorithm in MATLAB codes. The core bit count step is implemented as a mex function using SSE4 *popcnt* instruction. All experiments are run with Intel Xeon 3.6GHz x4 64bits CPU and 12GB RAM.

This supervised approach leads to significant performance improvement as compared to random projection adopted in LSH. Figure 1 illustrates improvements in precision rates measured on CIFAR10 dataset with both Euclidean neighbour and semantic labels. The improvements can also be found in Table 4 for experiments with other datasets. Figure 2 (left) illustrates how the precision of JSD algorithm is positively related to the number of candidate hash functions L. Whereas a small number of 200 candidates lead to significant improvement over random projections in LSH, 50 times more candidates improves about 10% precision only.

Precisions of JSD-based learning essentially depend on qualities of label information. Figure 3 shows that there is a graceful precision loss due to noisy or partial labels. With 50% random labels, the precision of JSD learning decreases about 20% yet still outperforming LSH. Training with 1% labels, the incurred precision loss is no more than 5%.

4.3. JSD improvements of other methods

Algorithm 1 can also be used to improve other binary code learning methods with a minor modification, i.e. by appending outputs of existing methods $\mathbf{H}_e \in \{0,1\}^{N \times K}$ to candidate projection outcomes $\mathbf{H} \leftarrow \mathbf{H} \cup \mathbf{H}_e$. This generic approach of exploiting label information can be applied to any hash coding learning algorithms in a consistent manner.

We set L (=10000+K) for all experiments reported in this paper. Figures 4 illustrates performance improvements of one unsupervised (SH [26]) and three supervised methods (BRE [7], ITQ [3] and KSH [12]) measured on CIFAR10 dataset with Euclidean labels. It is shown that JSD-based learning approaches, denoted as X-JSD, consistently improve all existing methods to various extents, with the only exception that precision-recall curves are almost identical for KSH and KSH-JSD.

Quantitative performance comparison, in terms of mean average precision (mAP), are summarized in Figure 5 in which six sets of data/labels are used. It is observed that performances of JSD-based learning approaches compare favourably with those of original methods in the majority of comparisons. While the average improvement of SH-JSD over SH is more noticeable (11.7%), the difference between KSH-JSD and KSH is merely 1.3%. Detailed comparison of mAP across different methods and datasets are illustrated in Table 4. Note that for datasets reported in top three panels, both the best scores (in bold font) and the runner ups (underlined) along each row bunch together in "KSH/JSD" column. In the bottom two panels, however, performances in KSH/JSD column no longer excel but JSD learning methods still outperform the original methods in the majority of cases (with minor exceptions marked by "‡"). This adaptation demonstrates that the JSD learning approach is not biased and can be effectively applied to any existing methods.

Note that certain columns of $\mathbf{H}_e$ might be selected if their JSD measures are the largest one at some steps. For in-

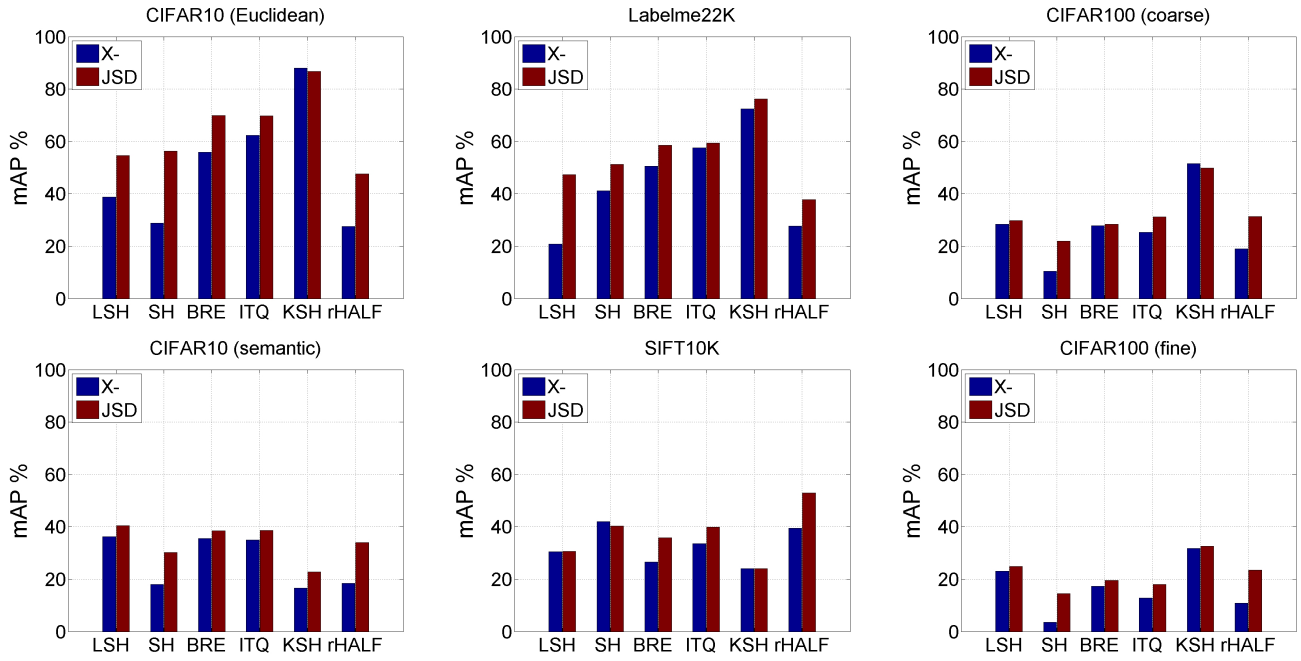

Figure 5. Comparison of "JSD" learning method with original methods (denoted by "X"). Mean average precision (mAP %) are measured only for Hamming radius no greater than 3. Six datasets are used and the rHALF method is introduced in Section 4.4.

K (bits)	8	12	16	24	32	48	64	selected
K_{SH}	3	4	4	5	9	27	42	38%
K_{BRE}	0	0	0	2	13	26	42	24%
K_{ITQ}	0	0	0	0	6	22	35	17%
K_{KSH}	7	7	12	18	27	42	60	80%

Table 1. Numbers and average percentages of hash functions that are selected from different methods.

L	200	600	1000	2000	6000	10000
T_p(s)	2.39	1.63	2.09	3.31	8.88	29.47
K	8	16	24	32	48	64
T_s(s)	6.6	56.9	111.1	116.5	121.2	129.9

Table 2. Summary of time complexity.

stance, Table 1 summarizes numbers of hash functions that are selected from different methods in one example run. In particular, the last column shows percentage of hash functions that are selected from corresponding learning methods, averaged over different lengths of hash codes. Noticeably, KSH contributes the majority of selected hash functions and this seems in accordance with high precision rates of KSH reported in Figure 4 as well as Table 4.

4.4. Time Complexity and Supervised Haar-like Functions

Table 2 summarizes average time costs of two time-consuming steps in the proposed JSD learning algorithm. First, the preparation time (T_p) of candidate matrix $\mathbf{H}$ is approximately proportional to the number of candidates L (also see Figure 2 right). Second, the training time (T_s) mainly depends on the number of bits K. However, T_s does not increase exponentially with K since the number of data points is finite and $p_i^{b_1 \ldots K}$ quickly approach 0 for many subsets after dozens of iterations. Therefore, a small probability threshold $p_t (= 0.0001)$ is used to early stop many branches whenever $p_i^{b_1 \ldots K} < p_t$. Precision loss incurred by early stopping is negligible.

Relatively long preparation time T_p and training time T_s is not a critical issue for off-line training, which is often the case for large scale image searching applications. Instead, the processing time of converting a query data point into binary codes (i.e. *coding time*) is a crucial performance parameter for nearest neighbour search algorithms. While the coding time is often considered a constant for projection based hash functions, it actually can be reduced by order of magnitudes by exploiting simple yet discriminative *binary tests* of input feature vectors.

Haar-Like Functions: Inspired by the well known Haar-like features for real-time object detection [22], we propose a set of Haar-Like Functions (HALF) as follows:

$$h_{ij}(\mathsf{x}) = \begin{cases} 1 & \text{if } x_i > x_j, i \neq j \\ 0 & \text{otherwise} \end{cases}, \qquad (7)$$

where x_i is i-th component of the input vector $\mathsf{x} = (x_1, x_2, ...x_p) \in \mathbb{R}^p$.

This family of hash functions constitute a special subset of the well-known linear projections that have only two non-zero elements (1 and -1 respectively) in each column

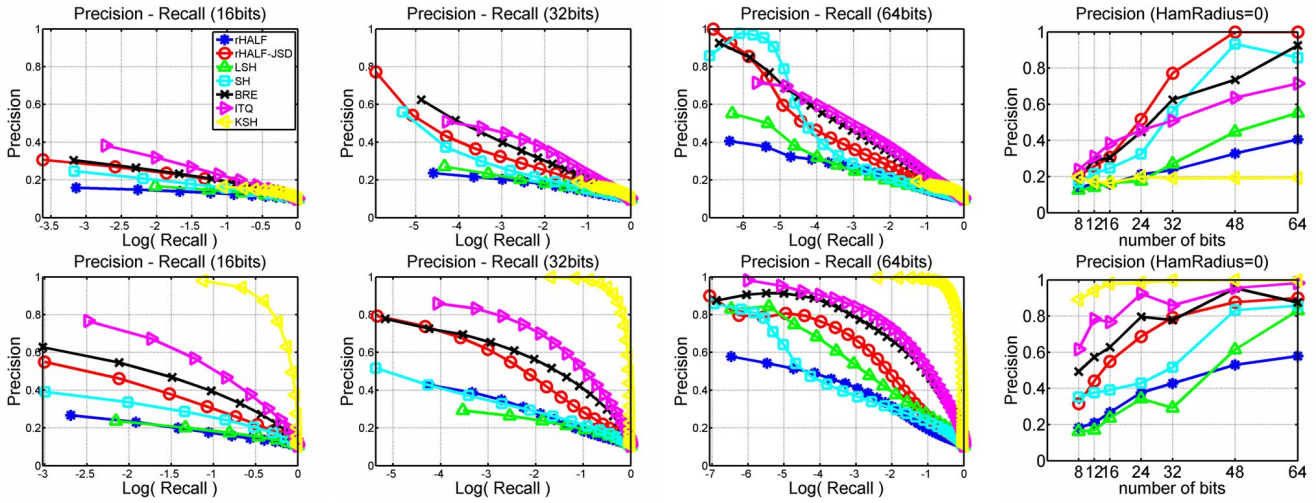

Figure 6. Performance comparison on CIFAR10 dataset with semantic labels (**upper row**) and Euclidean neighbourhood (**bottom row**). Three columns on left: Precision-recall curves with varying code lengths. Right-most column: summary of precisions.

of the projection matrix W. For p-dimensional input vectors, there are in total $\binom{p}{2}$ candidate HALFs from which K HALFs are to be selected. Random selection of HALFs (rHALF) does not necessarily guarantee satisfactory precision rates, instead, we adopt the proposed JSD learning algorithm to boost precision rates of random HALFs (denoted as rHALF-JSD hereafter).

It is observed in Figure 5 that rHALF-JSD significantly outperforms rHALF by about 14% improvement in mAP. As compared with other methods, Figure 6 shows that the curve of rHALF-JSD (red solid line) lies almost exactly between unsupervised (LSH, rHALF and SH) and supervised methods (BRE, ITQ and KSH). It is also shown in the right most column of Table 4 that rHALF-JSD outperforms all other methods for 128 dimensional SIFT feature vectors. Given that HALFs merely amount to a tiny fraction of all possible linear projections, its performance is acceptable.

The practical value of rHALF-JSD lies in its extremely short coding time T_c (see Table 3). Since there is no matrix multiplication involved, the per query coding time is at least two orders of magnitude faster than other projection based methods for 512-dimensional GIST vectors. This speedup is particularly useful for real-time applications such as fast keypoint recognition [16] and image localization [18] in which query images may contain up to 10K features and the coding time amounts to a significant portion of the total processing time. The advantage of the proposed Haar-Like function becomes more pronounced for applications running on mobile devices.

5. Conclusion

Contributions of this work are two-folds. First, we formulated binary hash coding learning within a statistical learning framework in which an upper bound of the probability of Bayes decision errors is derived for arbitrary hash functions. Since the convergence property of such a sequential learning method is rigorously proved, one is able to freely apply the JSD-based learning approach to different hash functions as long as sufficient number of labelled data are available. Our experimental results also demonstrated the viability of such a generic approach. Second, we proposed a very simple and fast hash function which is able to reduce the coding time by two order of magnitudes as compared with other projection based methods. Due to its high speed and computational simplicity, the proposed Haar-Like function is especially preferred for applications running on mobile devices.

As for future work, we advocate the JSD-based learning approach as a generic framework to combine different learning algorithms. In order to alleviate the burden of providing labels for all data points, we are interested in a semi-supervised approach where the second term of JSD (in eq. 3) can be estimated using only partially labelled data.

	T_c(s)	T_p(s)	T_s(s)
rHALF-JSD	1.0×10^{-7}	183.6	3337.5
LSH	80×10^{-7}		0.5
SH	400×10^{-7}		3.0
BRE	290×10^{-7}		494.7
KSH	430×10^{-7}		156

Table 3. Average time cost of rHALF-JSD in comparison with that of LSH, SH, BRE and KSH. T_c - coding time. T_p - preparation time. T_s - training time.

References

[1] A. Bosch, A. Zisserman, and X. Munoz. Image classification using random forests and ferns. In *ICCV*, 2007. 2

[2] M. Calonder, V. Lepetit, C. Strecha, and P. Fua. Brief: binary robust independent elementary features. ECCV'10, 2010. 1, 2

Table 4. Comparison of *mean average precision* i.e. *mAP* (%), for which only Hamming radiuses no great than 3 are considered. The highest rates along each row are in bold font, while the second best underlined. ‡ indicates those exceptional cases in which JSD does not outperform the original method. See discussion in Sections 4.3 and 4.4.

Dataset	Code Length	Approaches					
		LSH / JSD	SH / JSD	BRE / JSD	ITQ / JSD	KSH / JSD	rHALF / JSD
CIFAR10 (Euclidean)	8	23.86 / 33.05	13.78 / 29.51	30.73 / 48.87	37.44 / 47.56	**70.56** / **70.56**	14.02 / 20.55
	16	24.39 / 40.09	16.49 / 41.79	39.36 / 59.16	48.13 / 56.46	**84.86** / 80.43‡	18.19 / 31.52
	32	31.03 / 64.88	21.00 / 64.22	63.24 / 77.42	73.73 / 78.62	**97.18** / 96.56‡	30.52 / 58.61
	64	75.76 / 80.25	63.99 / 89.97	90.46 / 94.38	90.06 / 96.33	99.22 / **99.30**	47.62 / 79.79
Labelme22K	8	8.52 / 21.08	17.62 / 23.99	22.64 / 33.95	29.05 / 32.78	**48.07** / **48.07**	9.64 / 14.81
	16	10.77 / 34.30	19.73 / 31.76	33.51 / 44.15	39.21 / 41.45	57.99 / **68.84**	11.58 / 21.31
	32	22.27 / 56.78	32.08 / 53.89	56.23 / 64.90	69.82 / 67.92‡	84.96 / **89.63**	27.05 / 40.67
	64	41.84 / 77.21	95.49 / 95.57	89.94 / 91.49	92.16 / 95.45	**98.49** / 98.42‡	62.41 / 74.39
CIFAR100 (coarse)	8	6.01 / 6.06	5.41 / 6.24	6.57 / 6.87	6.57 / 7.04	**10.99** / **10.99**	5.46 / 6.39
	16	6.72 / 7.08	5.58 / 8.14	8.85 / 11.39	9.36 / 10.71	**23.85** / 19.64‡	5.83 / 8.73
	32	10.35 / 14.85	6.80 / 15.76	21.69 / 23.29	25.75 / 28.16	**79.91** / 74.52‡	8.48 / 18.57
	64	90.73 / 91.06	24.24 / 57.74	74.19 / 71.87‡	59.70 / 78.92	91.69 / **94.45**	56.04 / 91.52
CIFAR100 (fine)	8	1.44 / 1.56	1.12 / 1.49	1.72 / 1.85	1.70 / 1.81	**2.56** / **2.56**	1.18 / 1.50
	16	1.79 / 2.12	1.22 / 2.25	2.97 / 3.41	3.12 / 3.31	**5.67** / 4.75‡	1.53 / 2.16
	32	4.28 / 8.16	2.02 / 6.47	10.25 / 11.74	11.92 / 13.87	**44.86** / 41.08‡	2.49 / 8.06
	64	85.03 / **87.75**	10.18 / 48.00	54.27 / 61.35	34.72 / 53.05	73.89 / 81.82	38.50 / 82.20
CIFAR10 (semantic)	8	13.25 / 16.39	11.46 / 15.59	14.72 / 16.77	16.18 / 17.21	16.43 / **17.55**	11.67 / 15.20
	16	15.68 / 20.06	12.79 / 20.79	20.51 / 24.79	23.33 / **25.23**	15.20 / 20.40	13.09 / 20.27
	32	23.39 / 32.84	17.68 / 26.42	38.43 / 38.62	40.89 / **42.66**	17.10 / 25.34	18.76 / 34.52
	64	**92.94** / 92.34‡	30.12 / 57.83	68.63 / 73.81	59.66 / 69.60	18.04 / 27.78	30.27 / 66.01
SIFT10K	8	5.39 / 6.61	10.42 / 6.82‡	7.00 / 13.59	7.06 / 13.59	6.21 / 10.13	8.84 / **16.08**
	16	10.76 / 11.35	19.31 / 16.06‡	12.53 / 24.54	16.46 / 26.09	10.71 / 20.79	16.51 / **31.32**
	32	24.16 / 23.12‡	39.77 / 40.25	29.10 / 51.73	35.59 / 46.82	29.99 / 33.52	44.51 / **66.95**
	64	81.58 / 81.72	**98.66** / 98.13‡	57.63 / 53.65‡	75.21 / 73.36‡	49.11 / 31.80‡	87.88 / 97.20

[3] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. CVPR '11, 2011. 1, 5

[4] Y. Gong, S. Lazebnik, A. Gordo, and F. Perronnin. Iterative quantization: A procrustean approach to learning binary codes for large-scale image retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 99:1, 2012. 1

[5] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. STOC '98, 1998. 2, 3

[6] H. Jégou, M. Douze, C. Schmid, and P. Pérez. Aggregating local descriptors into a compact image representation. In *IEEE Transactions on Pattern Analysis and Machine Intelligence*. 1

[7] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *NIPS*, pages 1042–1050. 2009. 1, 2, 5

[8] V. Lepetit, P. Lagger, and P. Fua. Randomized Trees for Real-Time Keypoint Recognition. In *CVPR*, 2005. 2

[9] S. Leutenegger, M. Chli, and R. Siegwart. Brisk: Binary robust invariant scalable keypoints. In *ICCV'11*, pages 2548–2555, 2011. 1

[10] X. Li, G. Lin, C. Shen, A. van den Hengel, and A. Dick. Learning hash functions using column generation. In *International Conference on Machine Learning (ICML'13)*, Atlanta, USA, 2013. 2

[11] J. Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information theory*, 37:145–151, 1991. 3

[12] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *CVPR*, 2012. 1, 2, 3, 5

[13] M. Norouzi and D. J. Fleet. Minimal loss hashing for compact binary codes. In *ICML*, pages 353–360, 2011. 1, 2

[14] M. Norouzi, D. J. Fleet, and R. Salakhutdinov. Hamming distance metric learning. In *NIPS*, 2012. 1, 2

[15] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *Int. J. Comput. Vision*, 42(3):145–175, May 2001. 4

[16] M. Özuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(3):448–461, 2010. 1, 2, 7

[17] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski. Orb: An efficient alternative to sift or surf. In *ICCV*, 2011. 1, 2

[18] T. Sattler, B. Leibe, and L. Kobbelt. Fast image-based localization using direct 2d-to-3d matching. In *ICCV*, pages 667–674, 2011. 1, 7

[19] C. Strecha, A. M. Bronstein, M. M. Bronstein, and P. Fua. LDAHash: Improved Matching with Smaller Descriptors. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(1), 2012. 1

[20] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. In *CVPR*, 2008. 1

[21] T. Trzcinski and V. Lepetit. Efficient Discriminative Projections for Compact Binary Descriptors. In *ECCV*, 2012. 1, 2

[22] P. A. Viola and M. J. Jones. Rapid object detection using a boosted cascade of simple features. In *CVPR (1)*, pages 511–518, 2001. 2, 6

[23] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *CVPR*, 2010. 1, 2, 3

[24] J. Wang, S. Kumar, and S.-F. Chang. Sequential projection learning for hashing with compact codes. In *ICML*, 2010. 1, 2, 3

[25] J. Wang, J. Wang, N. Yu, and S. Li. Order preserving hashing for approximate nearest neighbor search. In *ACM*, 2013. 2

[26] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008. 1, 2, 5