Learning Binary Codes with Bagging PCA

Cong Leng[1], Jian Cheng[1,⋆], Ting Yuan[1], Xiao Bai[2], and Hanqing Lu[1]

[1] National Laboratory of Pattern Recognition, Institute of Automation, Chinese Academy of Sciences, Beijing, China
{cong.leng,jcheng,tyuan,luhq}@nlpr.ia.ac.cn
[2] School of Computer Science and Engineering, Beihang University, Beijing, China
baixiao@buaa.edu.cn

Abstract. For the eigendecomposition based hashing approaches, the information caught in different dimensions is unbalanced and most of them is typically contained in the top eigenvectors. This often leads to an unexpected phenomenon that longer code does not necessarily yield better performance. This paper attempts to leverage the bootstrap sampling idea and integrate it with PCA, resulting in a new projection method called Bagging PCA, in order to learn effective binary codes. Specifically, a small fraction of the training data is randomly sampled to learn the PCA directions each time and only the top eigenvectors are kept to generate one piece of short code. This process is repeated several times and the obtained short codes are concatenated into one piece of long code. By considering each piece of short code as a "super-bit", the whole process is closely connected with the core idea of LSH. Both theoretical and experimental analyses demonstrate the effectiveness of the proposed method.

Keywords: Bootstrap, random, bagging, PCA, binary codes, Hamming ranking.

1 Introduction

Hashing based approximate nearest neighbor (ANN) search is crucial for many large scale machine learning and computer vision tasks, such as image retrieval [24], object detection [5] and 3D reconstruction [23]. In these tasks, one is often required to find the nearest neighbor for one point in a huge database. Nearest neighbor (NN) search is unfeasible in these scenarios because its high time complexity. In the hashing based approaches, binary codes will be generated for points in the database and similar points will have close codes. Searching will be very fast because the Hamming distance between binary codes can be efficiently calculated with XOR instruction in modern CPU. Furthermore, the binary code can be very compact for memory storage.

The most well-known hashing method is Locality Sensitive Hashing (LSH) [13,4,3,20], which generates a batch of random projections to embed the data into Hamming space. Owing to the inner randomness, LSH based methods are *data-independent* and have nice theoretical properties. Indyk *et al.* [13] proved that two similar samples would be embedded to close codes with high probability, as long as the hash functions are of locality-sensitive function family. Moreover, they proved that the collision

⋆ Corresponding author.

T. Calders et al. (Eds.): ECML PKDD 2014, Part II, LNCS 8725, pp. 177–192, 2014.

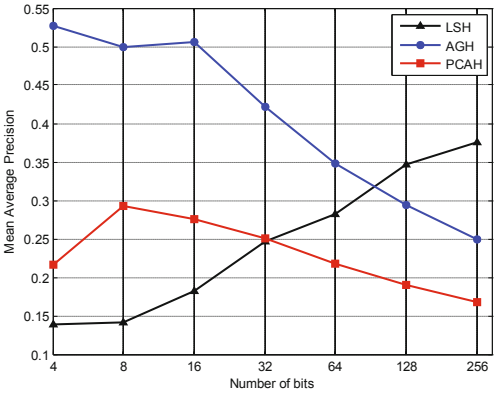

Fig. 1. Mean Average Precision of LSH, AGH and PCAH with various bits on toy data

probability would be higher as the code size increases [13]. However, in practice LSH typically needs very long codes and multiple tables to guarantee reasonable recall rate, which would degrade the search efficiency. By comparison, many recent hashing approaches attempt to learn data-aware hash functions by utilizing machine learning techniques. Several methods such as Spectral Hashing (SH) [26], Anchor Graph Hashing (AGH) [18], Iterative Quantization (ITQ) [7], Spherical Hashing (SpH) [11] and Kernel Supervised Hashing (KSH) [17] have been developed. These *data-dependent* methods aim to learn a set of projections, which are usually demonstrated to be more effective than the data-independent LSH.

From the mathematical perspective, lots of existing data-dependent hashing methods are based on eigendecomposition of matrix. Motivated by spectral clustering, SH [26] calculates the bits by thresholding a subset of eigenvectors of a Laplacian matrix of the similarity graph. AGH [18] follows the same idea of SH but utilizes anchor graph to overcome the computation problem in graph construction. Self-taught Hashing (STH) [29] first binarizes the eigenvectors of a normalized Laplacian and then learns SVMs as hash functions to overcome the out-of-sample extension problem. Other representative works include PCA Hashing (PCAH) [7], Semi-supervised Hashing (SSH) [24], Optimized Kernel Hashing (OKH) [8], etc.

For eigendecomposition based hashing algorithms, the information caught by different eigenvectors is unbalanced, that is, most of the information is typically contained in the top eigenvectors while the remainders are usually less informative or even noisy. Since each eigenvector is encoded into the same number of bits, the bits generated with the noisy eigenvectors will be noisy too, which will degrade the performance. As demonstrated in Fig.1, increasing number of bits leads to poorer mean average precision (MAP) performance on the toy data[1] with both PCAH and AGH. This is because more noise will be introduced while more eigenvectors are used. However, by intu-

[1] The used toy data is MNIST, which is available at `http://yann.lecun.com/exdb/mnist/`. 1,000 images are randomly selected as queries. The ground truth is defined as semantic neighbors based on digit labels.

ition, longer codes should catch more information than the shorter ones and give better retrieval performance, like that in LSH. On the other hand, although the short code generated with the top eigenvectors achieves decent performance, the data representation capability of short code is limited, and the most recently proposed hashing methods such as SpH [11] often use relatively longer code to get much better performance. Therefore, for eigendecomposition based methods, there exists a dilemma between the code length and performance.

Since most of information is contained in the top eigenvectors, one natural idea is to just use the top eigenvectors to generate a piece of short but strong code and repeat this process several times, and then concatenate these pieces of short codes into one piece of long code. However, it is obvious that if these pieces of short codes are identical, the obtained long code won't catch any more information. Thus, to get stronger long code, the short codes should be strong but also diverse. Inspired by the work of bagging strategy [2], in this paper, we adopt the widely used bootstrap technique [6] to generate diverse short codes. A new projection method, named Bagging PCA, is proposed and applied to learn effective binary codes. More specifically, each time we randomly sample a small subset of the training data to learn PCA directions, and only the top eigenvectors are kept to generate one piece of short code for all the data. This process will be repeated several times and afterwards the obtained many pieces of short codes will be concatenated into one piece of long code. The proposed hashing method, dubbed Bagging PCA Hashing (BPCAH), enjoys the following three appealing advantages:

- Since only the top eigenvectors are used every time, we can expect that the obtained binary codes would be more effective. Extensive experiments on three large scale datasets demonstrate that the proposed method outperforms several state-of-the-art hashing methods.
- It can be theoretically guaranteed that the longer codes tend to yield better results than the shorter ones under such a strategy. In addition, due to the randomness introduced by bagging, our method shares some common attributes with the well-known LSH, which is of nice theoretical properties.
- Because only PCA is used in the whole process, the proposed method is very suitable for large scale dataset. An important benefit of bagging scheme is that it is inherently favorable to parallel computing. Therefore, although the learning process has to be repeated several times, it can be completed in parallel with independent computation units.

2 Related Work

In this section, we give some backgrounds about hashing and introduce some related works which attempt to handle the unbalance problem in eigendecomposition based hashing methods. First of all, some notations are defined. Let $\mathcal{X} = \{x_1, x_2, \cdots, x_n\}$ denote a set of n data points, where $x_i \in \mathbb{R}^d$ is the ith data point. We denote $X = [x_1, x_2, \cdots, x_n] \in \mathbb{R}^{d \times n}$ as the data matrix. The binary code matrix of these points is $H = [h_1, h_2, \cdots, h_r] \in \{-1, 1\}^{n \times r}$, where r is the code length. Hashing code for one point is a row of H and denoted as $code(x_i)$.

As one of the most popular hashing methods, LSH randomly samples hash functions from a locality sensitive function family [1]. SimHash [4,13] and MinHash [3,20] are two widely adopted LSH schemes. MinHash is a technique for quickly calculating the Jaccard coefficient of two sets by estimating the resemblance similarity defined over binary vercotrs [3]. In contrast, SimHash is an LSH for the similarities (e.g., cosine similarity) which work on general real-valued data. As indicated in [21], when the data are high-dimensional and binary, MinHash tends to work better than SimHash. On the other hand, SimHash achieves better performance than MinHash on real-valued data. Specifically, to approximate the cosine similarity, Charikar [4] defined a hash function h as:

$$h(q) = \begin{cases} 1, & \text{if } w \cdot q > 0 \\ 0, & \text{if } w \cdot q < 0 \end{cases} \tag{1}$$

where w is a random vector from the d-dimensional Gaussian distribution $N(0, I_d)$. Although with abundant nice theoretical properties, these random projection based data-independent hashing methods are less discriminative over data and typically need very long codes to achieve satisfactory search performance.

Recently, many data-dependent hashing methods [26,18,15,11,24,29,16,7,9] have been proposed to learn data aware hash functions. As we have mentioned, many of them [26,18,24,29,8,7] are based on eigendecomposition of a matrix (e.g. Laplacian matrix). This brings the unbalance problem because the information caught in different eigenvectors is unbalanced. A few recent works have been proposed to address this problem.

In [25], instead of learning all the eigenvectors at once, Wang *et al.* proposed a sequential learning framework (USPLH) to learn hash function which tends to minimize the errors made by the previous one. Inspired by multiclass spectral clustering [28], in Iterative Quantization (ITQ) [7], Gong *et al.* proposed an alternating minimization scheme to learn an orthogonal transformation to the PCA projected data so as to minimize the quantization error of mapping the data to their corresponding binary codes (the vertices of binary hypercube). In Isotropic Hashing (IsoH) [15], Kong *et al.* proposed to learn projection functions which can produce projected dimensions with equal variances. Same as in ITQ, they tried to learn an orthogonal transformation to the PCA projected data by iteratively minimizing the reconstruction error of the covariance matrix and a diagonal matrix. Similar idea was adopted in [27], but in which the PCA projection was replaced with locality preserving projection (LPP) [10]. In these methods, longer codes often catch much more information and thus give better experimental results than the shorter ones. However, to the best of our knowledge, there still lack enough theoretical guarantee that the performance will be better as the size of codes increases, like in LSH.

The differences between our method and the previous works are obvious. Instead of minimizing the quantization error or toughly requiring each dimension to have equal variance, we leverage the bootstrap sampling scheme and integrate it with PCA. Every time only the informative top eigenvectors are used to learn short binary code. Owing to the sophisticated theories established in ensemble learning, our method enjoys several advantages which are lacking from previous works.

3 The Proposed Approach

Assuming the data X is zero-centered, for a projection $W = (w_1, w_2, \cdots, w_r) \in \mathbb{R}^{d\times r}$, code matrix can be written as $H = sgn(X^T W)$, where $sgn(\cdot)$ is the sign function. In general, for a code to be efficient, two requirements should be satisfied [26]: (1) each bit has a 50% chance of being +1 or -1, i.e. $\sum_i h_k(x_i) = 0, k = 1, 2, \cdots, r$; (2) different bits are independent of each other.

For the first requirement, Wang *et al.* [24] have proved that constraint $\sum_i h_k(x_i) = 0$ is equivalent to maximizing the variance for the k-th bit. The second "independent" requirement is often relaxed into the pairwise decorrelation of bits [26], i.e. $\frac{1}{n}H^T H = I$. In [24], it was further relaxed into the orthogonality constraints on the projection directions, i.e. $W^T W = I$. By dropping the non-differentiable $sgn(\cdot)$ function and trivially using $tr(AB) = tr(BA)$, overall, the objective can be formally written as [24,7,15]:

$$\begin{aligned} \underset{W\in\mathbb{R}^{d\times r}}{max} \quad & \frac{1}{n} tr(W^T X X^T W) \\ s.t. \quad & W^T W = I_r \end{aligned} \tag{2}$$

This objective function is exactly the same as that of Principal Component Analysis (PCA). The optimized projection W can be obtained by solving the top r eigenvectors corresponding to the r biggest eigenvalues of the data covariance matrix XX^T.

3.1 Hashing with Bagging PCA

For the eigendecomposition based hashing method, e.g. PCAH, on the one hand, the amount of information caught in different dimensions differs significantly, but each dimension is encoded into the same bits of code. This brings the unbalance problem in the obtained binary codes. On the other hand, the most discriminative information is typically contained in the top eigenvectors so that the short code generated with only the top eigenvectors often yield better retrieval performance than longer ones in these methods. In spite of this, the data representation capability of short codes is limited, and the most recently proposed hashing methods often use relatively longer code to achieve better performance.

We notice that the bagging strategy can enhance advantages and avoid disadvantages of the eigendecomposition based method. Bagging [2] is a classical and efficient combining technique for improving weak classifiers, and is extensively applied to classification problems [22]. Bagging is based on bootstrap [6] and aggregating concepts, and incorporates the benefits of both approaches. Bootstrap is based on random sampling in all the training data with replacement. Taking a bootstrap replicate of p samples $X^{(i)} = [x_1^{(i)}, x_2^{(i)}, \cdots, x_p^{(i)}]$ from the whole training set $\mathcal{X} = \{x_1, x_2, \cdots, x_n\}$, one can sometimes avoid or get less misleading training samples in the bootstrap training set. Aggregating actually means combining the base weak learners, and the combined learner typically gives better results than individual base learner. In general, bagging is helpful to build a better learner on training sets with misleading samples.

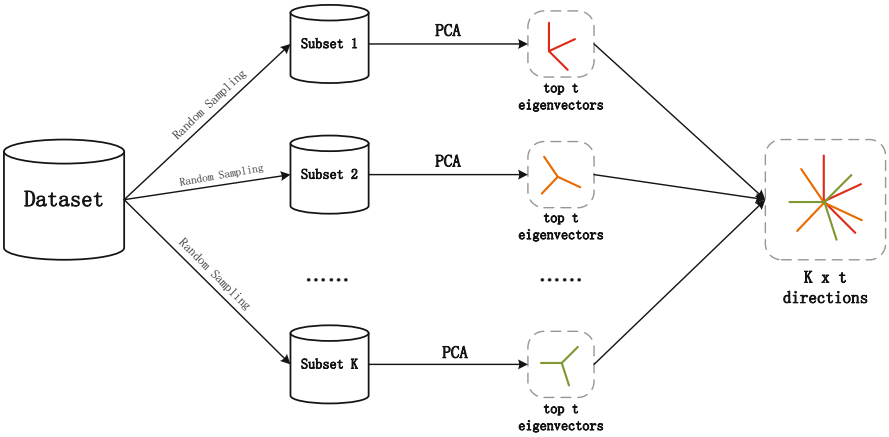

Fig. 2. The flowchart of Bagging PCA. At first, K bootstrap training set are randomly sampled from the whole training set. For each bootstrap, PCA is applied and only the top t eigenvectors (directions) are kept. Afterwards, the K diverse blocks of PCA directions are combined.

The motivation is clear: since most of the information is caught in the top eigenvectors, why not repeat using them to generate short codes and then concatenate them into long code. However, it is obvious that if the many pieces of short codes are identical, the obtained long code will not get any more information. Based on this idea, we propose a new projection method, named as Bagging PCA, and apply it to learn discriminative binary codes.

In the first step, a small subset of p samples $X^{(i)} = [x_1^{(i)}, x_2^{(i)}, \cdots, x_p^{(i)}]$ is randomly sampled from the training set X. With this bootstrap training set, we can learn the corresponding PCA projections and only the top t eigenvectors are kept. In other words, we optimize the following objective and get the optimized $W^{(i)} \in \mathbb{R}^{d \times t}$.

$$\begin{aligned} \underset{W^{(i)} \in \mathbb{R}^{d \times t}}{max} \quad & \frac{1}{p} tr({W^{(i)}}^T X^{(i)} {X^{(i)}}^T W^{(i)}) \\ s.t. \quad & {W^{(i)}}^T W^{(i)} = I_t \end{aligned} \tag{3}$$

With the sampling and learning process repeated K times, we can obtain K diverse blocks of PCA directions, i.e. $\{W^{(i)}\}_{i=1}^{K}$. The final projection matrix is generated by combining the K blocks:

$$W = \left[W^{(1)}, W^{(2)}, \cdots, W^{(K)}\right] \in \mathbb{R}^{d \times Kt} \tag{4}$$

The binary code for each data x can be written as $code(x) = sgn(x^T W)$. Note that this equals to concatenate many pieces of short codes into one piece of long code because $sgn(x^T W) = [sgn(x^T W^{(1)}), sgn(x^T W^{(2)}), \cdots, sgn(x^T W^{(K)})]$.

In some extreme cases, there exists heavy unbalance even between the top eigenvectors. In these circumstances, a random rotation can be applied to the learned $W^{(i)}$.

Algorithm 1. Hashing with Bagging PCA

Input: Training set $\mathcal{X} = \{x_1, x_2, \cdots, x_n\}$, the number of piece of short codes K, the number of samples in each bootstrap replicate p, the code size of short code t.

Output: Hashing codes of $K \times t$ bits as well as $K \times t$ hash functions.

1: Generate K bootstrap replicates $\{X^{(i)}\}_{i=1}^{K}$. Each replicate $X^{(i)}$ contains p training samples randomly selected from the whole training set $\mathcal{X}$ with n samples.
2: **for** $i = 1, \cdots, K$ **do**
3: Calculate the covariance matrix $C^{(i)} = X^{(i)} {X^{(i)}}^T$.
4: Get the t top eigenvectors $\{e_k^{(i)}\}_{k=1}^{t}$ of $C^{(i)}$, denoted as $W^{(i)} = [e_1^{(i)}, e_2^{(i)}, \cdots, e_t^{(i)}]$.
5: Optional: Apply a random rotation to $W^{(i)}$: $W^{(i)} \longleftarrow W^{(i)}R$.
6: **Coding:** for j = 1, $\cdots$, n, do
7: $code^{(i)}(x_j) \longleftarrow sgn(x_j^T W^{(i)})$
8: **end for**
9: Concatenate the K pieces of short codes $\{code^{(i)}\}_{i=1}^{K}$ of each sample into one piece of $(K \times t)$ bits binary code $[code^{(1)}, code^{(2)}, \cdots, code^{(K)}]$.

Obviously, if $W^{(i)}$ is an optimal solution of Eq.(3), then so is $W^{(i)}R$ for any $t \times t$ orthogonal matrix R. As indicated in [14], a random rotation is helpful for balancing the information in top eigenvectors. The flowchart of Bagging PCA is shown in Fig.2, and the proposed strategy for binary codes learning can be summarized as in Algorithm 1.

3.2 Theoretical Analysis

In the last decades, mature theory frame has been established in ensemble learning to guarantee correctness of the algorithms such as bagging, random forest and boosting. In this subsection, we generalize the discussion to our bagging PCA hashing. In specific, we intend to theoretically guarantee that longer hashing codes tend to give better retrieval performance in our method.

Let $f : \mathcal{X} \times \mathcal{X} \rightarrow [0, 1]$ be a ground truth similarity function over a set of objects $\mathcal{X}$, where we can interpret $f(x, y)$ to mean that x and y are "similar" or "dissimilar". The essence of hashing is constructing codes such that the Hamming distance between $code(x)$ and $code(y)$ corresponds to the ground truth similarity $f(x, y)$, i.e. two similar items have a small Hamming distance, while two dissimilar items have a large Hamming distance. Although discussing the Hamming distance, as indicated in [17], the Hamming distance $d(x, y)$ between $code(x)$ and $code(y)$ can be directly converted to a similarity measure $S(x, y)$ in Hamming space:

$$S(x, y) = \frac{r - d(x, y)}{r} \tag{5}$$

where r is the code length. Obviously, $d(x, y) \in [0, r], S(x, y) \in [0, 1]$, and smaller Hamming distance corresponds to larger similarity. From this perspective, we can consider that the essence of hashing is constructing codes for the data whose similarities match the ground truth as better as possible.

Lemma 1: *Concatenating K pieces of short codes of t bits into one piece of $K \times t$ bits long code, then the similarity between two samples evaluated with the long code is the mean of those evaluated with the K pieces of short codes.*

Proof: Denote the Hamming distance and similarity between two samples evaluated with the short codes as $d_i(x, y)$ and $S_i(x, y)$, where $i = 1, 2, \cdots, K$. Similarly, denote that evaluated with the long code (obtained by concatenating short ones) as $d_l(x, y)$ and $S_l(x, y)$, respectively. It is obvious that $d_l(x, y) = \sum_{i=1}^{K} d_i(x, y)$. Then we have

$$\begin{aligned} S_l(x,y) &= \frac{K \times t - d_l(x,y)}{K \times t} \\ &= \frac{1}{K}\left(\frac{K \times t - \sum_{i=1}^{K} d_i(x,y)}{t}\right) \\ &= \frac{1}{K}\left(\sum_{i=1}^{K}\left(\frac{t - d_i(x,y)}{t}\right)\right) \\ &= \frac{1}{K}\sum_{i=1}^{K} S_i(x,y) \end{aligned}$$

Theorem 1: *Under the bootstrap sampling framework, the similarity between two samples evaluated with the long code (obtained by concatenating short codes) tend to be closer to the ground truth than that evaluated with the short code.*

Proof: As shown in Lemma 1, the aggregated similarity evaluated with long code is:

$$S_l(x, y) = \mathbf{E}[S_i(x, y)] \tag{6}$$

It is easy to find that:

$$\begin{aligned} \mathbf{E}[(f(x,y) - S_i(x,y))^2] = f^2(x,y) - 2f(x,y)\mathbf{E}[S_i(x,y)] \\ + \mathbf{E}[S_i^2(x,y)] \end{aligned} \tag{7}$$

Since $E[Z^2] \geq (E[Z])^2$, we have

$$\mathbf{E}[S_i^2(x, y)] \geq (\mathbf{E}[S_i(x, y)])^2 \tag{8}$$

Eq.(7) can derive

$$\mathbf{E}[(f(x, y) - S_i(x, y))^2] \geq (f(x, y) - \mathbf{E}[S_i(x, y)])^2 \tag{9}$$

Plugging in Eq.(6), we have

$$(f(x, y) - S_l(x, y))^2 \leq \mathbf{E}[(f(x, y) - S_i(x, y))^2] \tag{10}$$

From Eq.(10) we can get some insights on how the longer code improve the ranking performance. The deviation of $S_l(x, y)$ to the true similarity $f(x, y)$ is smaller than that of $S_i(x, y)$ averaged over the bootstrap sampling distribution. As we have mentioned, the essence of hashing is constructing codes whose similarity can match the ground truth as better as possible. Therefore, we can draw the conclusion that longer codes tend to give better retrieval performance under such a strategy.

How much improvement we can get depends on how unequal the Eq.(8) is. The effect of diversity is clear. If $S_i(x, y)$ does not change too much with different i the two sides will be nearly equal, and bagging will not help. As an extreme example, if every time all the samples are used to train the model, $S_i(x, y)$ will be identical for different i, so the left side and right side of Eq.8 (so Eq.10) will be the same. But S_l is always not inferior to S_i in theory.

Bagging strategy has been proved to be an efficient way to reduce generalization error by combining results from multiple base classifiers. According to Hoeffding inequality [12], when the base classifiers are mutually independent, the generalization error of the ensemble reduces exponentially to the ensemble size, and ultimately approaches to zero as the ensemble size approaches to infinity. Similar theory can be applied in our approach, but here the generalization error is the deviation to the true similarity and the ensemble size is the length of code.

3.3 Connection with LSH

For LSH, Indyk *et al.* [13] have proved that two similar samples will be embedded into close codes with high probability and this probability will increase as the code size increases [13]. Actually, as pointed out in [7], LSH is guaranteed to yield exact Euclidean neighbors in the limit of infinitely many bits. As we have proved in Theorem 1, the obtained long code by concatenating short codes will result in smaller deviation to the true similarity, which is the same as in LSH.

On the other hand, if we treat one piece of short code in our method as a "super-bit", our method can be seen as a special case of LSH. The difference is that in LSH the hash functions are randomly generated but in our method we introduce randomness via randomly sampling the training data and every "super-bit" here is learned with consideration of the data. Our method enjoys the benefits of both data-independent and data-dependent methods.

3.4 Computation Complexity Analysis

There exists a straightforward question for Bagging PCA hashing, i.e. whether the bagging strategy will increase the computational complexity. The time complexity of PCA is $O(nd^2 + d^3)$, which is linear to the size of dataset. For the proposed BPCAH, each time the size of training set is p, which is much smaller than n. In addition, an important benefit of bagging scheme for hashing is that it is inherently favorable to parallel computing. With this benefit, although we have to repeat the learning process K times, it can be completed in parallel with K computation units. This characteristic is important for large scale dataset in real applications.

4 Experiments

To be consistent with the motivation and theoretical analysis given above, what we want to verify in experiments is threefold: (1) The proposed solution addressing the unbalance problem of eigendecomposition based methods outperforms other existing solutions. (2) Longer codes, as indicated in the previous section, do give better performance

Table 1. Description of Datasets

	CIFAR10	Tiny100K	GIST1M
Dimensionality	512	384	960
Size	60,000	100,000	1,000,000

than the shorter ones in the proposed method. (3) The proposed method outperforms other state-of-the-art hashing methods.

4.1 Experimental Setting

Datasets: Three large scale datasets are employed to evaluate the proposed method: CIFAR10, Tiny100K and GIST1M. CIFAR10[2] consists of 60K 32×32 color images in 10 classes, with 6000 images per class. We extract 512 dimensional GIST descriptor [19] to represent each image. Tiny100K consists of 100K tiny images randomly sampled from the original 80 million tiny images[3]. Each image is represented by 384 dimensional GIST descriptor. GIST1M contains one million 960 dimensional GIST descriptors extracted from random images. It is directly downloaded from the website[4]. We summarize the size and dimensionality of the datasets in Table 1. For each dataset, we randomly select 1,000 data points as queries and use the rest as gallery database and training set. Following [24,25], the top 2 percentile nearest neighbors in Euclidean space are taken as ground truth.

Compared Methods: We compare the proposed BPCAH with several state-of-the-arts hashing methods, including Locality Sensitive Hashing (LSH) [4], PCA Hashing (PCAH) [24], Anchor Graph Hashing (AGH) [18], Unsupervised Sequential Projection Learning Hashing (USPLH) [25], Iterative Quantization (ITQ) [7], Isotropic Hashing (IsoH) [15] and Spherical Hashing (SpH) [11]. As we have pointed out above, there are many versions of LSH. Here we adopt the algorithm proposed in [4] because the experimental data is dense and real-valued here. AGH is a popular eigendecomposition based hashing method. USPLH, ITQ, IsoH and our BPCAH are all based on PCA and aim to handle the unbalance problem in PCAH. SpH is a recently proposed hashing method which achieves promising retrieval performance on many datasets. We implement LSH and our method by ourselves, and use the source codes provided by the authors for all the other methods. In our method, there are two parameters to be set, the size of each bootstrap training set p and code length of each short code t. We set $p = 30\% \times n$ and $t = 16$ for all the comparisons.

Evaluation Criterions: To perform fair evaluation, we adopt the Hamming Ranking search commonly used in the literature. All points in the database are ranked according

[2] `http://www.cs.toronto.edu/~kriz/cifar.html`
[3] `http://groups.csail.mit.edu/vision/TinyImages/`
[4] `http://corpus-texmex.irisa.fr/`

to their Hamming distance to the query and the top K samples will be returned. The retrieval performance is measured with three widely used metrics: mean average precision (MAP), precision of the top K returned examples (Precision@K) and precision-recall curves. The MAP score is calculated by

$$MAP = \frac{1}{|Q|}\sum_{i=1}^{|Q|}\frac{1}{n_i}\sum_{k=1}^{n_i} precision(R_{ik})$$

where $q_i \in Q$ is a query, n_i is the number of points relevant to q_i in the dataset. Suppose the relevant points are ordered as $\{r_1, r_2, \cdots, r_{n_i}\}$, then R_{ik} is the set of ranked retrieval results from the top result until you get to point r_k.

4.2 Experimental Results and Analysis

MAP Scores: MAP is one of the most comprehensive criterions to evaluate the retrieval performance in the literature [25,7,15,11]. Table 2-4 show the MAP scores for all the algorithms on the three datasets. We observe that BPCAH achieves the highest MAP scores with different code lengths on all the datasets. Comparing the data dependent methods with the data-independent LSH, it can be observed that the data dependent methods like ITQ and SpH are generally better than LSH, especially with small code size. However, LSH results in higher MAP score as the code size increasing, for example, from 0.0946 (32 bits) to 0.2997 (256 bits) on CIFAR10. This behavior is due to the theoretical convergence guarantee of LSH that when enough bits are assigned, two similar samples will be embedded into close codes with high probability. By comparison, as the code size increases, the MAP scores of PCAH and AGH decrease. When the code size exceeds 64, the MAP score of AGH is even lower than LSH on all the datasets.

By comparing the MAP scores of our BPCAH with those of three other methods which also aim to address the unbalance problem of eigendecomposition based method, i.e. USPLH, ITQ and IsoH, we find that our method outperforms them with a large margin. This improvement is mainly due to that we only use the informative top eigenvectors and drop the noisy eigenvectors in learning each piece of short code in our method. These results imply that the proposed strategy of concatenating short codes generated with only the top eigenvectors into long code is very effective to handle the unbalance problem.

Considering the MAP scores of BPCAH with different code sizes, it is easy to find that our method yields to higher MAP score as the code size increases on all the datasets. The improvement from 32 bits to 64 bits is prominent, and becomes stable when the code size exceeds 128. This phenomenon is natural and easy to understand since in our method as the code size increases more useful information is integrated. From this perspective, our method is very similar to LSH. This verifies the claims made in the previous section, and it is an important characteristic of our method. Furthermore, our BPCAH consistently performs better than SpH, although the advantage is not so significant when the code size exceeds 256. SpH achieves promising performance on these datasets in the literature [11] and is the best competitor under most settings in our experiments. Given a MAP of 0.32 in Table 2, BPCAH needs to use about 48 bits to encode

Table 2. Mean Average Precision (MAP) scores for CIFAR10 dataset

Methods	Mean Average Precision					
	32-bits	48-bits	64-bits	96-bits	128-bits	256-bits
LSH	0.0946	0.1349	0.1591	0.2046	0.2344	0.2997
PCAH	0.0593	0.0596	0.0587	0.0567	0.0539	0.0464
AGH	0.1488	0.1528	0.1544	0.1511	0.1461	0.1290
USPLH	0.1048	0.1215	0.1213	0.1199	0.1196	0.1192
ITQ	0.2345	0.2604	0.2781	0.2999	0.3131	0.3450
IsoH	0.2138	0.2426	0.2612	0.2876	0.3058	0.3414
SpH	0.1745	0.2131	0.2422	0.2853	0.3198	0.3823
BPCAH	**0.2614**	**0.3170**	**0.3469**	**0.3679**	**0.3819**	**0.3926**

Table 3. Mean Average Precision (MAP) scores for Tiny100K dataset

Methods	Mean Average Precision					
	32-bits	48-bits	64-bits	96-bits	128-bits	256-bits
LSH	0.1280	0.1467	0.1746	0.2100	0.2401	0.2825
PCAH	0.0822	0.0770	0.0712	0.0635	0.0582	0.0460
AGH	0.1582	0.1621	0.1617	0.1547	0.1473	0.1292
USPLH	0.1240	0.1230	0.1229	0.1225	0.1222	0.1218
ITQ	0.2076	0.2294	0.2426	0.2593	0.2627	0.2821
IsoH	0.2008	0.2307	0.2428	0.2627	0.2726	0.2941
SpH	0.1872	0.2256	0.2463	0.2805	0.3066	0.3501
BPCAH	**0.2239**	**0.3040**	**0.3413**	**0.3669**	**0.3810**	**0.3930**

Table 4. Mean Average Precision (MAP) scores for GIST1M dataset

Methods	Mean Average Precision					
	32-bits	48-bits	64-bits	96-bits	128-bits	256-bits
LSH	0.0994	0.1242	0.1381	0.1697	0.1952	0.2453
PCAH	0.1088	0.0998	0.0914	0.0791	0.0718	0.0523
AGH	0.1346	0.1415	0.1455	0.1464	0.1460	0.1655
USPLH	0.1014	0.1006	0.1005	0.0999	0.0995	0.0990
ITQ	0.1878	0.2070	0.2188	0.2307	0.2383	0.2514
IsoH	0.1821	0.1999	0.2180	0.2334	0.2406	0.2592
SpH	0.1544	0.1946	0.2137	0.2447	0.2635	0.3082
BPCAH	**0.1961**	**0.2341**	**0.2517**	**0.2788**	**0.2951**	**0.3187**

each image while the best competitor SpH needs about 128 bits. In consequence, our method typically provides about three times more compact binary representation than other methods. Similar trends can be found in Table 3 and Table 4.

Precision@K: In some applications, what we really concern is the precision of the top K returned samples. For example, in real image retrieval system, most of the users only care about the returned images in the first page. Fig.3(a)-(c) show the precision of

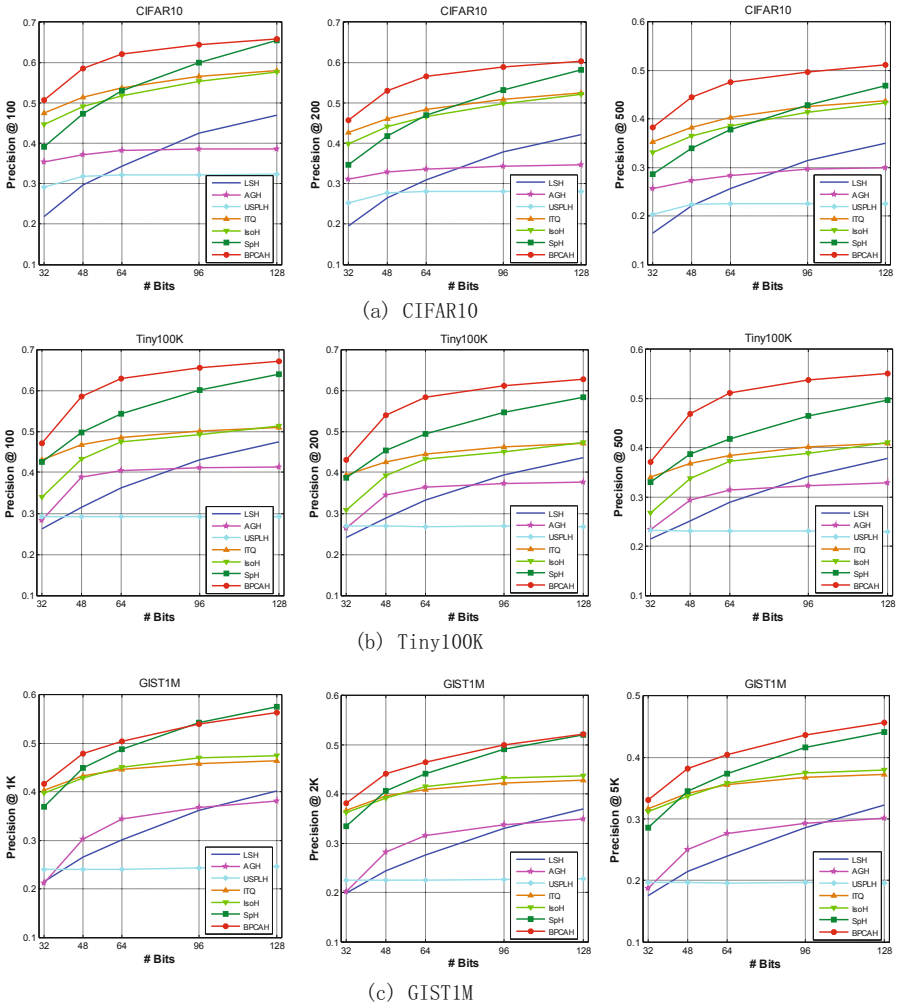

Fig. 3. Precision of top K returned of different methods on three datasets. (a) Precision@100,200,500 with various code sizes on CIFAR10. (b) Precision@100,200,500 with various code sizes on Tiny100K. (c) Precision@1K,2K,5K with various code sizes on GIST1M.

top K returned with different bits on the three datasets. For CIFAR10 and Tiny100K, precision on top 100, 200 and 500 is reported. For GIST1M, since the relevant samples for each query are more (top 2 percentile nearest neighbors are defined to be relevant), we report the precision on the top 1K, 2K and 5K. The performance of PCAH is not even comparable with other competitors as the code size exceeds 32. To avoid clutter, these curves and the subsequent results reported in this section omit the baseline method PCAH.

Our BPCAH consistently outperforms its competitors almost on all the cases. Once again, we gain remarkable improvement over USPLH, ITQ and IsoH. For instance, when the top 100 samples are returned in CIFAR10, our method achieves a precision

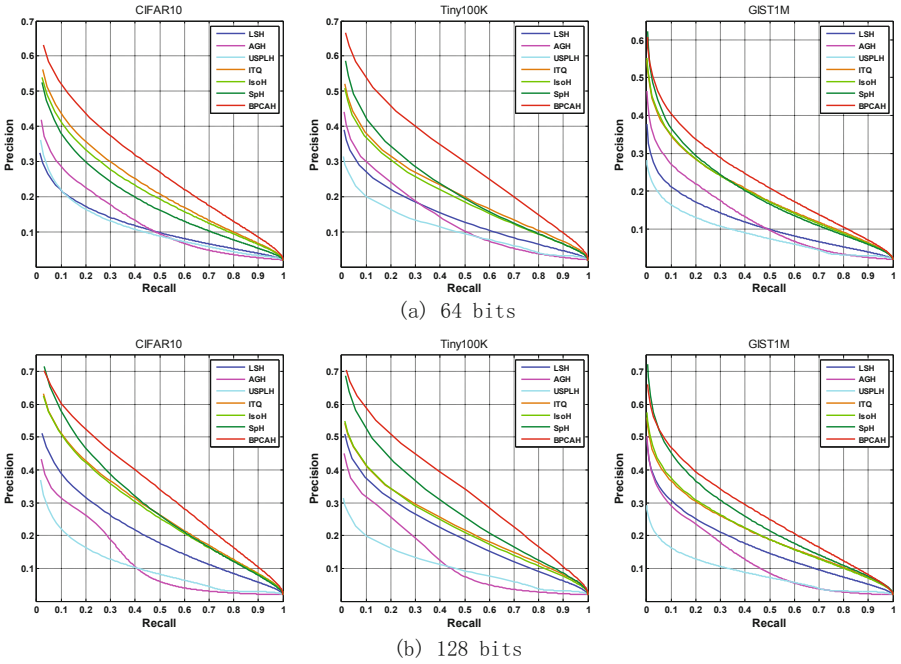

Fig. 4. (a) Precision-Recall curves with 64 bits on three datasets. (b) Precision-Recall curves with 64 bits on three datasets.

of 62% and the best competitor ITQ only arrives at 54%. We can also make some interesting observations about the performance of the other methods. In Fig.3(a)(c), ITQ and IsoH work relatively well for small code size and are better than SpH. However,

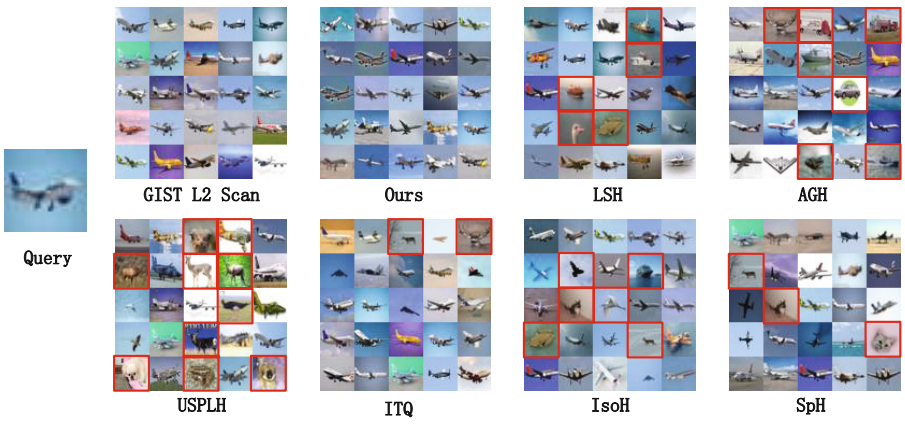

Fig. 5. Retrieval results on CIFAR10 using original gist descriptor, and binary codes build with different hashing methods. Top 25 returned are shown. We used 64-bit hashing codes, and show the false positives in red rectangle.

as the code size increases to 64, the performance of SpH rises rapidly and outperforms ITQ and IsoH (even ours in the left subfig of Fig.3(c)). As indicated in [11], this may be because the closed regions created by the hyperspheres is tighter than those created by the hyperplanes when multiple hyperspheres are used.

Precision-Recall Curves: Fig.4(a)(b) show the complete precision-recall curves on the three datasets with 64 bits and 128 bits. These detailed results are consistent with the trends that we discovered in the Table 2-4. Actually, MAP score is the area under the precision-recall curve. Fig.4 clearly shows the superiority of our BPCAH over other hashing methods.

In order to give an intuitive understanding about how these hashing methods work, Fig.5 shows an example with CIFAR10. An input query image on the left with 25 nearest neighbors using the original gist descriptor and binary codes built with different hashing methods are shown.

5 Conclusion and Future Work

In this paper, we proposed a new projection method named Bagging PCA for binary codes learning. The key idea is concatenating many pieces of diverse short codes into one piece of long code. In order to obtain diverse short codes, we adopted the bootstrap technique to learn PCA directions. Theoretical analysis and the connection with LSH were given. Extensive experiments on three large scale datasets demonstrated that our approach can outperform other state-of-the-arts hashing methods. Future work will explore the effectiveness of the proposed BPCAH on high-dimensional binary data such as text.

Acknowledgements. This work was supported in part by 973 Program (Grant No. 2010CB327905), National Natural Science Foundation of China (Grant No. 61170127, 61332016).

References

1. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: Proceeding of the Annual IEEE Symposium on Foundations of Computer Science (2006)
2. Breiman, L.: Bagging predictors. Machine Learning 24(2), 123–140 (1996)
3. Broder, A.Z., Charikar, M., Frieze, A.M., Mitzenmacher, M.: Min-wise independent permutations. In: Proceedings of the Annual ACM Symposium on Theory of Computing (1998)
4. Charikar, M.: Similarity estimation techniques from rounding algorithm. In: ACM Symposium on Theory of Computing, pp. 380–388 (2002)
5. Dean, T., Ruzon, M.A., Segal, M., Shlens, J., Vijayanarasimhan, S., Yagnik, J.: Fast, accurate detection of 100,000 object classes on a single machine. In: IEEE Conference on Computer Vision and Pattern Recognition (2013)
6. Efron, B., Tibshirani, R.: An introduction to the bootstrap, vol. 57. CRC press (1993)
7. Gong, Y., Lazebnik, S.: Iterative quantization: A procrustean approach to learning binary codes. In: IEEE Conference on Computer Vision and Pattern Recognition (2011)

8. He, J., Liu, W., Chang, S.F.: Scalable similarity search with optimized kernel hashing. In: Proceedings of the ACM SIGKDD Conference (2010)
9. He, K., Wen, F., Sun, J.: K-means hashing: an affinity-preserving quantization method for learning binary compact codes. In: IEEE Conference on Computer Vision and Pattern Recognition (2013)
10. He, X., Niyogi, P.: Locality preserving projections. In: Advances in Neural Information Processing Systems (2003)
11. Heo, J., Lee, Y., He, J., Chang, S., Yoon, S.: Spherical hashing. In: IEEE Conference on Computer Vision and Pattern Recognition (2012)
12. Hoeffding, W.: Probability inequalities for sums of bounded random variables. Journal of the American Statistical Association 58(301), 13–30 (1963)
13. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of ACM Symposium on Theory of Computing (1998)
14. Jégou, H., Douze, M., Schmid, C., Pérez, P.: Aggregating local descriptors into a compact image representation. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 3304–3311. IEEE (2010)
15. Kong, W., Li, W.: Isotropic hashing. In: Advances in Neural Information Processing Systems (2012)
16. Leng, C., Cheng, J., Lu, H.: Random subspace for binary codes learning in large scale image retrieval. In: Proceedings of ACM SIGIR Conference, SIGIR (2014)
17. Liu, W., Wang, J., Ji, R., Jiang, Y., Chang, S.: Supervised hashing with kernels. In: IEEE Conference on Computer Vision and Pattern Recognition (2012)
18. Liu, W., Wang, J., Kumar, S., Chang, S.: Hashing with graphs. In: Proceedings of the International Conference on Machine Learning (2011)
19. Oliva, A., Torralba, A.: Modeling the shape of the scene: A holistic representation of the spatial envelope. International Journal of Computer Vision 42(3), 145–175 (2001)
20. Shrivastava, A., Li, P.: Fast near neighbor search in high-dimensional binary data. In: Flach, P.A., De Bie, T., Cristianini, N. (eds.) ECML PKDD 2012, Part I. LNCS, vol. 7523, pp. 474–489. Springer, Heidelberg (2012)
21. Shrivastava, A., Li, P.: In defense of minhash over simhash. In: Proceedings of International Conference on Artificial Intelligence and Statistics (2014)
22. Skurichina, M., Duin, R.P.: Bagging, boosting and the random subspace method for linear classifiers. Pattern Analysis & Applications 5(2), 121–135 (2002)
23. Strecha, C., Bronstein, A.M., Bronstein, M.M., Fua, P.: Ldahash: Improved matching with smaller descriptors. IEEE Transactions on Pattern Analysis and Machine Intelligence 34(1), 66–78 (2012)
24. Wang, J., Kumar, S., Chang, S.F.: Semi-supervised hashing for scalable image retrieval. In: IEEE Conference on Computer Vision and Pattern Recognition (2010)
25. Wang, J., Kumar, S., Chang, S.F.: Sequential projection learning for hashing with compact codes. In: Proceedings of International Conference on Machine Learning, pp. 1127–1134 (2010)
26. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: Advances in Neural Information Processing Systems (2008)
27. Xu, B., Bu, J., Lin, Y., Chen, C., He, X., Cai, D.: Harmonious hashing. In: Proceedings of International Joint Conference on Artificial Intelligence (2013)
28. Yu, S.X., Shi, J.: Multiclass spectral clustering. In: Proceedings of the International Conference on Computer Vision (2003)
29. Zhang, D., Wang, J., Cai, D., Lu, J.: Self-taught hashing for fast similarity search. In: Proceedings of International ACM SIGIR Conference (2010)