Semi-supervised Nonlinear Hashing Using Bootstrap Sequential Projection Learning

Chenxia Wu, Jianke Zhu, Deng Cai, Chun Chen, and Jiajun Bu

Abstract—In this paper, we study the effective semi-supervised hashing method under the framework of regularized learning-based hashing. A nonlinear hash function is introduced to capture the underlying relationship among data points. Thus, the dimensionality of the matrix for computation is not only independent from the dimensionality of the original data space but also much smaller than the one using linear hash function. To effectively deal with the error accumulated during converting the real-value embeddings into the binary code after relaxation, we propose a semi-supervised nonlinear hashing algorithm using bootstrap sequential projection learning which effectively corrects the errors by taking into account of all the previous learned bits holistically without incurring the extra computational overhead. Experimental results on the six benchmark datasets demonstrate that the presented method outperforms the state-of-the-art hashing algorithms at a large margin.

Index Terms—Hashing, semi-supervised hashing, nearest neighbor search.

1 Introduction

Due to the excessive cost in storage and processing, an explosive growth of multimedia contents on the Internet brings both challenges and opportunities for knowledge discovery. Among these methods fast nearest neighbor search (or similarity search) has been a key step in many large-scale computer vision and information retrieval tasks, such as content-based multimedia retrieval [17], [27], [14], [2], near-duplicate detection [11], plagiarism analysis [26], collaborative filtering [15], and caching [22].

The fast nearest neighbor search methods can be roughly divided into two categories. One focused on finding the appropriate spatial partitions of the feature space via various tree structures including k-d trees [7], [24], M-trees [4], cover trees [1], metric trees [30], etc. These techniques attempt to speed up the nearest neighbor search. However, they are typically quite memory-demanding, which limits their capability of dealing with the large-scale data. Moreover, the searching performance drops significantly on the data with high dimensionality. In the worst case, the tree-based techniques are degenerated into a linear scan. Another category is the hashing-based method [3], [12], [5], [13], [9], [23], [34], [28], [16], [10], [37], [31], [32], [19], [35], [36], which has recently attracted considerable attention to achieve the fast query time while substantially reduce the storage requirement. In this paper, we focus on the learning-based hashing methods.

- *Corresponding author: Jianke Zhu is with College of Computer Science, Zhejiang University, Hangzhou, 310027, China.*
E-mail: `jkzhu@zju.edu.cn`.
Chenxia Wu, Chun Chen and Jiajun Bu are with College of Computer Science, Zhejiang University.
Deng Cai is with the State Key Lab of CAD&CG, College of Computer Science, Zhejiang University, Hangzhou, Zhejiang, China, 310058.
Email: `dengcai@cad.zju.edu.cn`.

Hashing can be viewed as a task of mapping high-dimensional input data into a low-dimensional binary space while appropriately preserving the distances or semantic relationship in the original feature space. Unlike the conventional dimensionality reduction techniques, the binary embedding is critical to assure the fast retrieval performance. One can perform efficient linear scans of the binary data to rank the objects in database according to their Hamming distances to the query, which is known as Hamming ranking. The Hamming distance between two objects can be computed within a few milliseconds on an ordinary PC today, which only involves an XOR operation along with a bit count. Even a linear scan in the Hamming space to rank the objects to a query can be performed very quickly. If the input dimensionality is very high, hashing methods lead to enormous computational savings.

Usually, we just need to retrieve a small number of the most similar objects, i.e., nearest neighbors, for a given query rather than computing its similarity to all objects in the database. In such situation, one can simply return all the objects that are hashed into a tight Hamming ball centered around the binary code of the query by Hash lookup. Also, it will be easy to filter or re-rank the very small set of "good" objects returned by hash lookup based on their full content so as to further improve the retrieval performance with just a little extra time [25].

The unsupervised hashing methods, such as Locality-Sensitive Hashing (LSH) [3], [12], Spectral Hashing (SH) [34], Anchor Graph Hashing (AGH-2) [19], usually try to directly employ the data information to compute the binary hashing code, which is difficult to preserve the semantic similarity due to the lack of the label information. It is possible for many applications to obtain a few data points with labels. Recently, several semi-supervised hashing methods [31], [32] are proposed to take advantage of the information of both labeled and unlabeled data.

Digital Object Indentifier 10.1109/TKDE.2012.76 1041-4347/12/$31.00 © 2012 IEEE

In the case of semi-supervised hashing[31], [32], the hash function is usually formulated as a linear projection coupled with the mean thresholding firstly. Secondly, the hash functions are derived according to both labeled and unlabeled data after the relaxation that the binary constraints for embeddings are removed. Finally, the real value embeddings are converted into binary one to make final solutions. Therefore, the generalization capability of the pre-assumed hash function at the first step, the objective function utilizing both labeled and unlabeled data at the second step and how to reduce the error accumulated for converting the real-value embeddings into the binary code at the third step (which is called Hashing Accumulated Error (HAE)) , will work together to determine the hashing performance.

Although some promising results demonstrate the effectiveness of existing semi-supervised hashing approaches[31], [32], there still exist two major issues: 1) the hash function is formulated as a linear projection coupled with the mean thresholding. The limited linear hash function may not effectively reflect the underlying relationship among the data points. Moreover, the linear formulation will involve with computation on the matrix with high dimensionality, which will incur the heavy computational cost in the case of the data with high dimensionality. 2) To deal with HAE, the previous methods [34], [32], [19] are lack of an appropriate error correction scheme, which should take consideration of all the embeddings holistically.

To address the above limitations, in this paper, we present a semi-supervised nonlinear hashing method using Bootstrap Sequential Projection Learning (Bootstrap-NSPLH).

Firstly, we employ a nonlinear one to improve the the generalization capability in contrast to the linear hashing function in the conventional method. To efficiently derive the embeddings, we take advantage of an anchor graph-based nonlinear hash function. Since such nonlinear mapping function is independent from the dimensionality of the original data space, which makes it practicable for the data of high dimensionality. Note that the nonlinear mapping can be viewed as a sparse representation that tries to reconstruct data point through a convex combination of its closest *anchors*, which is able to effectively capture the underlying geometry information among the data points.

Secondly, the presented method takes advantage of a regularized learning-based objective function of nonlinear hashing to maximize both the empirical accuracy and the information provided by each bits, which can utilize both the labeled and unlabeled data.

Finally, instead of the boosting-like process in [32], we propose a novel bootstrap-style sequential learning scheme to derive the hash function by correcting the errors in consideration of all previous learned bits holistically without incurring extra computational overhead. To effectively reduce the HAE, the pairwise label matrix is updated by imposing higher weights on point pairs violated by all previous learned hash functions.

We have conducted the extensive experiments on six benchmark datasets to evaluate the efficacy of the proposed method against several state-of-the-art hashing algorithms. The experimental results demonstrate that the nonlinear hash function performs much better than the linear one and the proposed bootstrap learning scheme achieves the best performance comparing to other learning schemes.

The remainder of this paper is organized as follows. In section 2, we introduce the related work on learning-based hashing. Section 3 presents the proposed semi-supervised nonlinear hashing formulation. Section 4 describes the details of our proposed bootstrap sequential projection learning for semi-supervised hashing. The computational complexity is analyzed in Section 5. Section 6 provides extensive experimental results on the six benchmark datasets. Section 7 gives the conclusion and future work.

We use the following notations in this paper. A training dataset of n d-dimensional points is denoted as $X = [\mathbf{x}_1, \cdots, \mathbf{x}_n] \in R^{d\times n}$, among which the label information of a small fraction is known. According to the type of pairwise label relationship, each pair of labeled points is categorized into two groups: positive group (P), in which each pair of samples shares the same label; negative group (N), in which each pair of samples belongs to the different classes. Suppose there are l points, $l < n$, each of which is associated with either P or N. Moreover, the matrix formed by these columns of X is denoted as $X_l \in R^{d\times l}$. The m-bit Hamming embeddings of X and X_l are denoted as $Y \in \{1, -1\}^{m\times n}$ and $Y_l \in \{1, -1\}^{m\times l}$ whose columns are the Hamming embeddings of corresponding points in X and X_l. E represents the expectation.

2 Related Work

Considerable research efforts have been devoted to the hashing-based method in machine learning and data mining community [3], [18], [32]. The existing work on learning-based hashing can be generally classified into two categories: unsupervised hashing and semi-supervised hashing.

2.1 Unsupervised Hashing

Most previous studies on hashing are unsupervised, which try to directly capture the data information for extracting the binary hashing code. Among these methods, locality-sensitive hashing (LSH) [3], [12] is a popular hashing method with many extensions for accommodating the distances such as l_p norm [5], learned metric [13], and image kernels [9]. Though LSH guarantees the desirable collision probability, it is not very efficient in practice due to involving with the multiple tables with long codes [8].

Several recent methods have explored to improve upon the random projection techniques used in LSH. Spectral Hashing (SH) is a popular data-dependent hashing [34], which assumes the data points in $\mathcal{X}$ had a uniform distribution. Specifically, PCA transformation on the data points is computed followed by the calculation of the top m smallest single eigenfunctions of the Laplacian, the sign of which yields the binary code. Anchor Graph Hashing (AGH-2) is an unsupervised graph-based method [18] which automatically discovers the neighborhood structure inherent in the data to learn the appropriate compact codes.

Usually, it is difficult for the unsupervised hashing methods to preserve the semantic similarity due to the lack of the label information while many applications can obtain a few data points with label information.

2.2 Semi-Supervised Hashing

Most recently, several semi-supervised hashing methods [31], [32] are proposed to take advantage of the information of both labeled and unlabeled data.

Generally, the objective of hashing is to find a compact representation for data in Hamming space where the Hamming distance between two binary codes accurately capture the semantic relationship of the corresponding data points:

$$\min E\{d_{H^m}(\mathbf{y}_i, \mathbf{y}_j)|P\} - E\{d_{H^m}(\mathbf{y}_i, \mathbf{y}_j)|N\} \tag{1}$$

The Hamming distance $d_{H^m}(.)$ is typically defined as

$$d_{H^m}(\mathbf{y}_i, \mathbf{y}_j) = \frac{m}{2} - \frac{1}{2}\sum_{k=1}^{m} \mathbf{y}_{ki}\mathbf{y}_{kj} \tag{2}$$

which is computed by counting the total number of non-zero bits in the XOR results between $\mathbf{y}_i$ and $\mathbf{y}_j$.

Removing the constant and replacing the expectation with the summation, the objective function in Eqn. 1 is rewritten into the following compact matrix form:

$$\begin{array}{c} \max \frac{1}{2}\mathbf{tr}\{Y_l S Y_l^\top\} \\ \text{s. t. } Y_l \in \{1, -1\}^{m\times l} \end{array} \tag{3}$$

where S is a pairwise label matrix, and each element S_{ij} is calculated by:

$$S_{ij} = \begin{cases} 1 & : \text{ if } \quad (\mathbf{x}_i, \mathbf{x}_j)\in P \\ -1 & : \text{ if } \quad (\mathbf{x}_i, \mathbf{x}_j)\in N \\ 0 & : \text{ otherwise.} \end{cases} \tag{4}$$

The above objective function can be employed to faithfully derive the Hamming embeddings for the data points in the training set according to their label information. For many real-world applications, however, it is typically expensive and tedious to annotate the data. Therefore, the labeled data usually account for a small portion of the whole dataset, i.e., $l \ll n$. Hence, directly solving the maximization problem in Eqn. 3 may lead to the overfitting problem.

2.3 Semi-Supervised Linear Hashing

The hash function for the semi-supervised hashing method [31], [32] is composed of a linear projection coupled with the mean thresholding scheme:

$$\mathbf{y_k} = \mathbf{sgn}(\mathbf{w_k}^\top \mathbf{x} - t_k) \tag{5}$$

where $\|\mathbf{w_k}\|^2 = 1$. $t_k = \frac{1}{n}\sum_i^n \mathbf{w_k}^\top \mathbf{x_i}$, which is set to zero after normalizing the training data with zero mean. Thus, the concise hash function could be obtained:

$$\mathbf{y_k} = \mathbf{sgn}(\mathbf{w_k}^\top \mathbf{x}).$$

Similarly, $\mathbf{sgn}(.)$ is removed when deriving the sequence of projection vectors $W = [\mathbf{w_1}, \ldots, \mathbf{w_m}] \in R^{d\times m}$ to obtain final hash function.

After this relaxation [31], [32], semi-supervised linear hashing tries to maximize the following objective function:

$$\max_W \frac{1}{2}\mathbf{tr}\{W^\top X_l S X_l^\top W\} + \frac{\lambda}{2}\mathbf{tr}\{W^\top X X^\top W\} \tag{6}$$

where $\mathbf{tr}\{W^\top X_l S X_l^\top W\}$ is according to the empirical accuracy using label information. $\mathbf{tr}\{W^\top X X^\top W\}$ is the regularizer to maximize the information provided by each bit, which can deal with the overfitting problem in Eqn. 3.

The maximization problem in Eqn. 6 for the linear hash functions is directly solved by retaining the top m eigenvectors of $M = X_l S X_l^\top + \lambda X X^\top$ as W and thresholding the linear projection $\mathbf{w_k}^\top \mathbf{x}$ to retrieve the binary code, which is then named as PCA hashing (PCAH) in [32].

Although the linear projection hash function obtains the promising results in some applications, there still remain two major challenges. One is the linear hash function may not effectively reflect the underlying relationship among the data points. The other is the computation for a $d \times d$ matrix may incur the heavy computational cost in the case of the data with high dimensionality.

2.4 Boosting Sequential Projection Learning

In [32], Wang et. al presented a boosting sequential projection learning hashing method (boosting-SPLH) for the semi-supervised hashing to deal with HAE. Their method is based on the objective function in Eqn. 6 for linear hashing. Boosting-SPLH corrects the errors in the previous dimensions so that W is learned iteratively. At each iteration, one projection $\mathbf{w}_k$ is learned by retaining the first eigenvectors (corresponding to the largest eigenvalue) of current M. Then the pairwise label matrix S is updated by assigning the higher weights to those point pairs violated the learned projection in this iteration:

$$S^{k+1} = S^1 - \tau \Delta S^k \tag{7}$$

$$\Delta S_{ij}^k = \begin{cases} E_{ij}^k & : \ S_{ij}^1 \cdot \mathbf{sgn}(E_{ij}^k) < 0 \\ 0 & : \ S_{ij}^1 \cdot \mathbf{sgn}(E_{ij}^k) \geq 0 \end{cases} \tag{8}$$

Here S^k represents the the pairwise label matrix S (Eqn. 4) for k-th projection learning. $E^k = X_l \mathbf{w}_k \mathbf{w}_k^\top X_l$ is simply the derivative of empirical accuracy of k-th hash function.

It can be observed that boosting-SPLH judges every previous bit separately when deciding the errors of learned projections in order to penalize with the higher weights. More specifically, boosting-SPLH only refers to the latest learned projection by E^k, considering that errors occur when the sign of single bit are different for positive pairs or are the same for negative pairs. This may incur more errors, since the hashing performance depends on how the binary representation similarity/distance accurately measures the semantic relationship of the corresponding data points which should be taken in consideration of all bits holistically. Additionally, since the dimension of M

changes with the dimension of data points, to obtain the top eigenvector of high dimension M becomes a bottleneck in computation when the original dimension of the data points is very high.

To deal with the limitations of previous methods, in this paper, we employ the more general nonlinear hash function instead of the linear one to effectively capture the underlying relationship among the data points. Moreover, a novel bootstrap sequential learning scheme is proposed to handle the hashing errors in the semi-supervised hashing.

3 Semi-Supervised Nonlinear Hashing

To address the limitations in the semi-supervised linear hashing, in this section, we present a semi-supervised nonlinear hashing method, in which a nonlinear projection coupled with mean thresholding employed as the hash function to effectively capture the underlying geometric information among the data.

As described in the recent work on large graph construction [18] and data-dependent hashing [19], the nonlinear hash function can be defined as follows:

$$\mathbf{y}_k = \mathbf{sgn}(\mathbf{w}_k^\top \mathbf{z}(\mathbf{x}) - t_k) \tag{9}$$

where k is the projection index, $\mathbf{w}_k$ is a projection vector with the magnitude $\|\mathbf{w}_k\|^2 = 1$, and t_k is a mean threshold, i.e., $t_k = \frac{1}{n}\sum_i^n \mathbf{w}_k^\top \mathbf{z}(\mathbf{x}_i)$. $\mathbf{z}(\mathbf{x})$ is the $\mathbf{x}$'s corresponding column of a regression matrix Z that measures the underlying relationship between the raw samples and the corresponding *anchors*, in which a small set of r points are employed to approximate the data neighborhood structure [19].

To compute the matrix Z, the clustering method, i.e., k-means, is firstly performed on n data points in order to obtain the $r(r \ll n)$ cluster centers $U = \{u_i \in R^d\}_i^r$ that act as *anchors*. Then, the Anchor Graph defines the truncated similarities Z_{ij}'s between r *anchors* and all n data points as:

$$Z_{ij} = \begin{cases} \frac{\mathbf{exp}(-D^2(\mathbf{u}_i,\mathbf{x}_j)/\theta)}{\sum_{i' \in <j>} \mathbf{exp}(-D^2(\mathbf{u}_{i'},\mathbf{x}_j)/\theta)}, \text{if} & i \in <j> \\ 0\ , & \text{otherwise.} \end{cases} \tag{10}$$

where $<j> \subset [1:r]$ denotes the indices of the $s(s \ll r)$ nearest *anchors* for the input $\mathbf{x}_j$ in U according to a distance function $D(\cdot)$, i.e., L_2 distance. θ denotes the bandwidth parameter.

We denote the non-zero elements of $\mathbf{z}(\mathbf{x})$ as $\hat{\mathbf{z}}(\mathbf{x})$, which can be viewed as the combination coefficients to reconstruct the data point $\mathbf{x}$ through a convex combination of its closest anchors by minimizing the reconstruction error [18]:

$$\min_{\hat{\mathbf{z}}(\mathbf{x}) \in R^s} \|\mathbf{x} - U_\mathbf{x}\hat{\mathbf{z}}(\mathbf{x})\|^2 \tag{11}$$
$$\text{s. t. } \mathbf{1}^\top \hat{\mathbf{z}}(\mathbf{x}) = 1, \hat{\mathbf{z}}(\mathbf{x}) > 0$$

where $U_\mathbf{x} \in R^{d\times s}$ is the sub-matrix of U composed of s nearest *anchors* of $\mathbf{x}$. Note that $\mathbf{z}(\mathbf{x})$ is able to effectively capture the underlying geometry information through kernel regression. Moreover, the dimension of $\mathbf{z}(\mathbf{x})$ is independent from the dimensionality of the original data space, which makes it practicable for the data of high dimensionality. In our empirical evaluation, we find that rather small r and s is sufficient to achieve the promising results.

Without loss of generality, let $\mathbf{z}(\mathbf{x})$ be normalized to have zero mean. Hence, t_k is set to zero for mean thresholding, and the following hash function is used for training:

$$\mathbf{y}_k = \mathbf{sgn}(\mathbf{w}_k^\top \mathbf{z}(\mathbf{x})) \tag{12}$$

By introducing the hash function using *anchors*, we can reformulate the original hashing problem defined in Eqn. 3 into the nonlinear form, which aims to derive the projection matrix $W = [\mathbf{w}_1, \dots, \mathbf{w}_m] \in R^{r\times m}$. Typically, a relaxation is introduced through removing the binary constraints, which leads to the following objective function:

$$\max_W \frac{1}{2}\mathbf{tr}\{W^\top Z_l S Z_l^\top W\} \tag{13}$$

where S is the similarity matrix in Eqn. 4, and $Z_l \in R^{r\times l}$ is the truncated similarity matrix defined in Eqn. 10 for the labeled data points $\mathbf{x} \in X_l$. To avoid the overfitting issue, a regularizer [32] is added to maximize the information provided by each bit, i.e., $W^\top ZZ^\top W$ where $Z \in R^{r\times n}$ is the truncated similarity matrix in Eqn. 10 for all data points $\mathbf{x} \in X$. Thus, we can obtain the objective function for the regularized nonlinear hashing as follows:

$$\max_W \frac{1}{2}\mathbf{tr}\{W^\top Z_l S Z_l^\top W\} + \frac{\lambda}{2}\mathbf{tr}\{W^\top ZZ^\top W\} \tag{14}$$

4 Bootstrap Sequential Projection Learning

In this section, we first define the Hashing Accumulated Error to show why a sequential learning scheme is introduced. Then, we present the proposed bootstrap sequential learning for semi-supervised hashing to effectively handle the Hamming accumulated errors problem.

4.1 Hashing Accumulated Error

Note that the maximization problem in Eqn. 14 for the nonlinear hash functions can be directly solved by retaining the top m eigenvectors of $Q = Z_l S Z_l^\top + \lambda ZZ^\top$ as W and thresholding the linear projection $\mathbf{w}_k^\top \mathbf{z}(\mathbf{x})$ to retrieve the binary code, which is named as nonlinear PCA hashing (NPCAH) in this paper.

However, directly thresholding the m functions as mentioned above is not equally suitable for hashing especially when m increases due to the error accumulates rapidly in converting the real-valued eigenvector $\mathbf{y}_k$ to the optimal integer solution $\mathbf{y}_k^* \in \{1, -1\}^m$ as k increases. Here, we name such error as Hashing Accumulated Error:

Definition 1 (Hashing Accumulated Error (HAE))**.** *To seek an m-bit Hamming embedding $Y \in \{1,-1\}^{m\times n}$, one derives the real-valued eigenvector $\mathbf{y}_k$ as the k-th Hamming embedding by removing the constraint $Y \in \{1,-1\}^{m\times n}$. As k increases, the error accumulated for converting $\mathbf{y}_k$*

into its optimal binary solution $\mathbf{y}_k^* \in \{1,-1\}^m$ is defined as Hashing Accumulated Error.

From the definition, we find that HAE causes the lower quality for the higher bits in hashing. In our empirical study (Section 6.4), we find that HAE would greatly affect the hashing performance when the bit length m increases to a small value, i.e., 16 bits. How to reduce HAE is the key to all eigen-based hashing methods using the relaxation that the binary constraints for embeddings are removed.

4.2 Bootstrap Sequential Projection Learning

In this paper, we propose a novel sequential projection learning scheme in consideration of all bits holistically without incurring any cost in contrast to the boosting one. To effectively capture the underlying relationship among the data points, we adopt the objective function in Eqn. 14 for the nonlinear hash functions. Consequently, the dimension of the matrix Q for computation is not only independent from the dimensionality of the original data space but also much smaller than the one using linear hash function. Hence, we name the presented hashing method as Bootstrap Sequential Projection Learning for Semi-supervised Nonlinear Hashing (Bootstrap-NSPLH).

The main difference between the boosting learning scheme [32] and the proposed bootstrap one is the error condition. The former judges every previous bit separately, i.e., errors occur when the sign of previous one bit is different for positive pairs or is the same for negative pairs. More precisely, the latter judges all previous bits holistically to decide the errors, i.e., errors occur in case of either the positive pairs with large Hamming distances or the negative pairs with small Hamming distances. Mathematically, "large" and "small" are able to be determined by two thresholds αk and βk respectively after k times projections. Thus, we formulate the error condition after k times projections as follows:

$$\begin{aligned}&\frac{k}{2} - \sum_{t=1}^{k} \mathbf{sgn}(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_i)\mathbf{z}(\mathbf{x}_j)^\top \mathbf{w}_t) > \alpha k, (\mathbf{x}_i,\mathbf{x}_j) \in P \\ &\qquad\text{or} \\ &\frac{k}{2} - \sum_{t=1}^{k} \mathbf{sgn}(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_i)\mathbf{z}(\mathbf{x}_j)^\top \mathbf{w}_t) < \beta k, (\mathbf{x}_i,\mathbf{x}_j) \in N \quad (15)\end{aligned}$$

Here $\frac{k}{2} - \sum_{t=1}^{k} \mathbf{sgn}(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_i)\mathbf{z}(\mathbf{x}_j)^\top \mathbf{w}_t)$ computes the Hamming distance as defined in Eqn. 2. We can remove the constants of above conditions to get a compact form:

$$\begin{aligned}&\sum_{t=1}^{k} \mathbf{sgn}(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_i)\mathbf{z}(\mathbf{x}_j)^\top \mathbf{w}_t) < \alpha k, (\mathbf{x}_i,\mathbf{x}_j) \in P \\ &\qquad\text{or} \\ &\sum_{t=1}^{k} \mathbf{sgn}(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_i)\mathbf{z}(\mathbf{x}_j)^\top \mathbf{w}_t) > \beta k, (\mathbf{x}_i,\mathbf{x}_j) \in N \quad (16)\end{aligned}$$

Here, $\sum_{t=1}^{k} \mathbf{sgn}(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_i)\mathbf{z}(\mathbf{x}_j)^\top \mathbf{w}_t)$ can be seen as the similarity between $\mathbf{x}_i$ and $\mathbf{x}_j$ in the binary metric space, whose value for k-bit can range from $-k$ to k increasing with the step size 2 i,e. $-k, -k+2, \ldots, k-2, k$. Thus, α and β are in the range from -1 to 1.

To represent the error condition, we store the similarities between each pair of data points into a matrix H^k with:

$$H_{ij}^k = \sum_{t=1}^{k} \mathbf{sgn}(\mathbf{w}_t^\top \mathbf{z}(\mathbf{x}_i)\mathbf{z}(\mathbf{x}_j)^\top \mathbf{w}_t)$$

which is further written into the matrix form:

$$H^k = \sum_{t=1}^{k} \mathbf{sgn}(Z_l^\top \mathbf{w}_t \mathbf{w}_t^\top Z_l) \quad (17)$$

Instead of calculating H^k with all the previous projection results, we compute it through an efficient recursive update equation below:

$$H^k = H^{k-1} + \mathbf{sgn}(Z_l^\top \mathbf{w}_k \mathbf{w}_k^\top Z_l) \quad (18)$$

Remark Note that we only need to calculate $\mathbf{sgn}(Z_l^\top \mathbf{w}_k \mathbf{w}_k^\top Z_l)$ at each iteration to obtain H^k. Thus, the time complexity for calculating H^k is $\mathcal{O}\left(l^2\right)$ rather than $\mathcal{O}\left(k \cdot l^2\right)$ in Eqn. 17. Therefore, the time complexity for updating of bootstrap scheme is $\mathcal{O}\left(m \cdot l^2\right)$, which is the same as boosting scheme in [32].

As the sign of the initial pairwise label matrix defined in Eqn. 4, S_{ij}^1 can represent the logical relationship of a point pair $(\mathbf{x}_i,\mathbf{x}_j)$: $(\mathbf{x}_i,\mathbf{x}_j) \in P$ or $(\mathbf{x}_i,\mathbf{x}_j) \in N$. The error conditions in Eqn. 16 can be rewritten with respect to S_{ij}^1 and H^k:

$$\begin{aligned}&H_{ij}^k - \alpha k < 0, S_{ij}^1 > 0 \\ &\qquad\text{or} \\ &H_{ij}^k - \beta k > 0, S_{ij}^1 < 0 \quad (19)\end{aligned}$$

According to error condition, the updating rule for S is formulated as follows:

$$S^{k+1} = S^1 + \Delta S^k \quad (20)$$

ΔS^k is the increased weight matrix which is determined by the error conditions:

$$\Delta S_{ij}^k = \begin{cases} (\alpha k - H_{ij}^k)/2k & : \ H_{ij}^k - \alpha k < 0, S_{ij}^1 > 0 \\ (\beta k - H_{ij}^k)/2k & : \ H_{ij}^k - \beta k > 0, S_{ij}^1 < 0 \\ 0 & : \ otherwise \end{cases} \quad (21)$$

Since $\alpha k - H_{ij}^k > 0$ and $\beta k - H_{ij}^k < 0$, the sign of S^{k+1} is the same as S^1 while its magnitude $|S_{ij}^{k+1}|$ becomes larger. $\alpha k - H_{ij}^k$ and $\beta k - H_{ij}^k$ are used to increase the weight when error occurs since it is proportional to the error. Extremely, the positive data pair with the largest Hamming distance or the negative pair with the smallest Hamming distance will be assigned with the largest weight. Both of them are divided by $2k$ to normalize the updating weight at each iteration ranging from -1 to 1.

After extracting the projection direction by Q^k for k-th bit, the contribution of its spanned subspace is removed from Z in order to minimize the redundancy in bits. Therefore, Z is updated as follows:

$$Z = Z - \mathbf{w}_k \mathbf{w}_k^\top Z = (I - \mathbf{w}_k \mathbf{w}_k^\top) Z \quad (22)$$

Algorithm 1 Bootstrap Sequential Projection Learning for Semi-supervised Nonlinear Hashing

```
**Input:** data $X$, pairwise labeled data $X_l$, initial pairwise labels $S^1$, length of hash codes $m$, constant $\lambda$;
**Output:** projection matrix $W$;
Computing $Z$, $Z_l$ using Eqn. 10;
Initialize $H^0 = 0$, $C^1 = ZZ^\top$;
**for** $k = 1$ to $m$ **do**
    Calculate the new adjusted matrix:
    $Q^k = Z_l S^k Z_l^\top + \lambda C^k$;
    Extract the first eigenvector $\mathbf{e}$ of $Q^k$ and set $\mathbf{w}_k = \mathbf{e}$;
    Calculate the pairwise Hamming similarity matrix:
    $H^k = H^{k-1} + \mathbf{sgn}(Z_l^\top \mathbf{w}_k \mathbf{w}_k^\top Z_l)$;
    Calculate the $\Delta S^k$ according to Eqn. 20;
    Calculate the new label pairwise matrix:
    $S^{k+1} = S^1 + \Delta S^k$;
    Calculate the residual:
    $U^k = I - \mathbf{w}_k \mathbf{w}_k^\top$, $C^{k+1} = U^k C^k {U^k}^\top$;
**end for**
**return** $W$
```

It is important to note that we can reduce the computational cost by calculating the update of $ZZ^\top$ recursively. Specifically, we denote $U^k = I - \mathbf{w}_k\mathbf{w}_k^\top$ and the covariance matrix at the first iteration as $C^1 = ZZ^\top$. According to Eqn. 22, we can efficiently calculate the covariance matrix C^{k+1} for the next iteration by the following equation:

$$C^{k+1} = U^k C^k (U^k)^\top \tag{23}$$

We can observe that directly calculating the term $ZZ^\top$ will involve with the matrix multiplication of high dimensional data matrix $Z \in R^{r\times n}(r \ll n)$. Since both C^k and U^k are with the size of $r \times r$, we can efficiently compute the residual by the recursive updating equation in Eqn. 23. The detailed procedure is summarized into Algorithm 1.

In addition, the proposed bootstrap sequential learning scheme can also be used to solve the problem with linear projection hash functions (Eqn. 6), we name it as bootstrap sequential projection learning hashing (Bootstrap-SPLH). For the boosting sequential learning with nonlinear projection hash functions, we name it as boosting nonlinear sequential projection learning hashing (Boosting-NSPLH).

5 Complexity Analysis

In this section, we study the computational complexity of the training process and the compression process for both the linear and the nonlinear hash function with either the boosting or the proposed bootstrap sequential learning scheme.

The cost in sequential projection learning is determined by two main steps: (1) Returning the top eigenvector; (2) Updating the pairwise label matrix. The cost in step (1) depends on the dimension of the matrix ($d \times d$ for the linear hash function and $r \times r$ for the nonlinear one) and the method used to compute the top eigenvector. From our empirical study in Section 6.3, we find that a small r is sufficient for leading performance compared with d. Thus, the cost for the nonlinear hash function is much smaller than that for the linear one in step (1).

Note that the boosting and bootstrap learning scheme share the same complexity at this step. The cost for the boosting learning at step (2) is $\mathcal{O}\left(m \cdot l^2\right)$ for l labeled data points. For the bootstrap one, calculating the pairwise Hamming similarity matrix and calculating the new label pairwise matrix in Algorithm 1 cost $\mathcal{O}\left(l^2\right)$. Also, the time complexity for the bootstrap learning scheme at step (2) is $\mathcal{O}\left(m \cdot l^2\right)$. Thus, the the computational complexity for the boosting and the proposed bootstrap sequential learning scheme are the same. The cost in this step is unrelated with the form of the hash function. In summary, the training process of our proposed Bootstrap-NSPLH costs the least among the four combinations.

The cost in the process of compressing the real value data into binary one is the key to the hash efficiency. To get m-bits binary embeddings for one d-dimensional data point costs $\mathcal{O}(m \cdot d)$ using the linear hash function and costs $\mathcal{O}(m \cdot s + d \cdot r)$ using the nonlinear one. In the experiments (Section 6.3), we find that very small r and s (set to 300 and 2 in our experiments) is sufficient to gain good performance, which preserves an efficient compression process for the nonlinear hash function. Since s is very small to preserve the sparsity of $z(x)$, the compression complexity for the nonlinear hash function hardly increases with the length of the bits. Moreover, one can tune the parameters r and s to tradeoff between the effectiveness and the efficiency for the nonlinear hash function. The cost in compression process is unrelated with the learning scheme.

6 Experiments

In this section, we present the details of our empirical studies on several benchmark datasets: MNIST, ISOLET, USPS, Caltech101, SIFT1M, PATCH1M. To evaluate the proposed Bootstrap-SPLH and Bootstrap-NSPLH method, we conduct an extensive comparison with several state-of-art approaches, including SH [34], PCAH [32], NPCAH, Boosting-SPLH [32], Boosting-NSPLH and AGH-2 [19]. For simplicity, “Boosting” and “Bootstrap” are denoted as “BS” and “BT” in our illustrations.

6.1 Datasets and Settings

Six benchmark datasets are employed to evaluate the proposed semi-supervised hashing methods. Specifically, we list our experiment settings in the following.

MNIST The MNIST handwritten digits database. The dataset consists of 784-dimensional $70K$ samples associated with digits from 0 to 9. We randomly split this dataset into two subsets: a training set containing $6.9K$ samples and a query set of $1K$ samples. In the case of semi-supervised hashing, the label information of $1K$ samples from the training set are used and the remaining data are treated as the unlabeled data. For the unsupervised case, all the training samples are treated as the unlabeled ones.

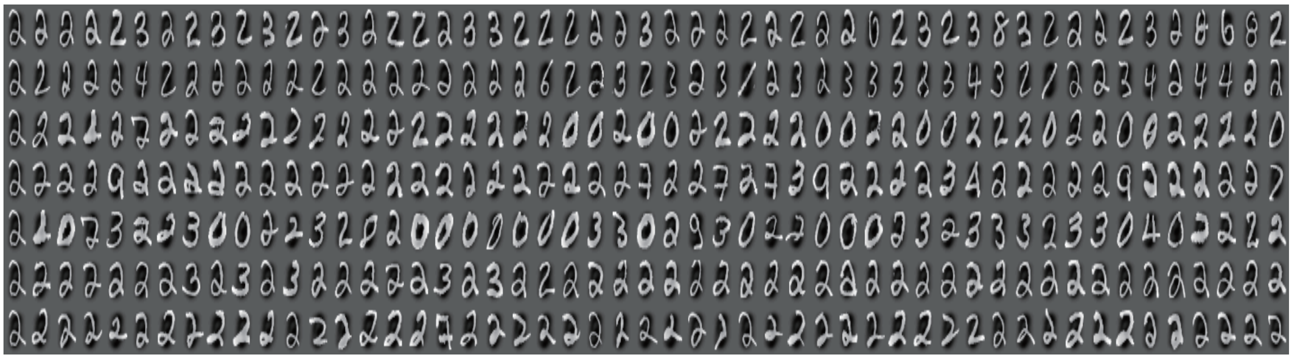


Fig. 1. The top 50 nearest neighbors of one example digit returned by the different methods on the MNIST dataset. The left most digit is the query sample. From top to bottom, the nearest neighbors are returned by SH, PCAH, NPCAH, Boosting-SPLH, Boosting-NSPLH, Bootstrap-SPLH, Bootstrap-NSPLH using 24-bit code.

TABLE 1
Datasets Description

	MNIST	ISOLET	USPS	Caltech101	SIFT1M	PATCH1M
data size	70K	8797	9298	9144	1M	1M
dimension	784	617	256	1024	128	256
classes	10	26	10	101	unknown	50

ISOLET The ISOLET spoken letter recognition database. The dataset was generated as follows. 150 subjects spoke the name of each letter of the alphabet twice. Hence, the dataset has 26 categories and $7,797$ samples. The dimension of each sample is 617. Similarly, we randomly partition the dataset into two parts: a training set with $6,797$ samples and a query set with $1K$ samples. For semi-supervised case, the label information of $1K$ samples from the training set are used and others are treated as the unlabeled data. For the unsupervised case, all the training samples are treated as the unlabeled ones.

USPS The USPS handwritten digit database. We choose a popular subset containing $9,298$ 16×16 handwritten digit images in total. Each sample is associated with a label from 0 to 9. We randomly partition the dataset into two parts: a training set with $8,298$ samples and a query set with $1K$ samples. For the semi-supervised case, the label information of $1K$ samples from the training set are used and others are treated as the unlabeled. In the case of unsupervised hashing, all the training samples are treated as the unlabeled ones.

Caltech101 The Caltech101 [6] contains the images of objects database. The dataset is a standard benchmark for object recognition in the vision community. $9,144$ images of objects belonging to 101 categories are in the database. About 40 to 800 images per category. Most categories have about 50 images. We adopt the method in [33] to obtain the $1,024$ dimensional BOW feature for every image. For the semi-supervised case, we randomly select 30 samples from each category, totally $3,030$ pictures, as the labeled training samples. The remainder samples are randomly split to $5,114$ unlabeled training samples and $1K$ query samples. In the case of unsupervised hashing, all $8,144$ training samples are treated as the unlabeled ones.

SIFT1M The SIFT1M is made of one million SIFT descriptors [20] extracted from random images, which is provided by the authors in [32]. Each item in the dataset is a 128-dimensional vector representing histograms of gradient orientations. As in [32], we employ one million samples for training and additional $10K$ as query samples, and Euclidean distance is used to determine the nearest neighbors. Following the criterion in [34], [31], a returned point is considered as a true neighbor if it lies in the top two percentile closest points to a query. In the semi-supervised methods, we randomly select $8K$ points from the training set and use the same criterion to determine the positive pairs. For the negative pair, a point belongs to the different class if it lies in the top two percentile farthest points.

PATCH1M We collected a large-scale dataset named as PATCH1M that contains one million labeled keypoint patches. We generate the database following the criterion from a learning-based keypoint recognition method [21]. Specifically, we first detect 50 stable keypoints in a model image and assign each keypoint a unique class number. The size of each keypoint patch is 16×16. A database of $1M$ keypoint patches for each class is built by generating thousands of sample images using the homographic matrices with randomly picked affine deformations by sampling the deformation parameters from a uniform distribution, adding Gaussian noise to each sample image, and smoothing with a Gaussian filter of size 7×7. We randomly split this dataset into two subsets: a training set containing $1M$ samples and a query set of $10K$ samples. In the case of semi-supervised hashing, the label information of $8K$ samples from the training set are used and the remaining data are treated as the unlabeled data.

The information of six datasets are summarized into Table 1. For all the nonlinear methods, we set the number of *anchors* r to 300, and s is equal to 2 for all six datasets. The selection of these two parameters will be discussed in Section 6.3. To make fair comparisons, all the nonlinear methods employ the same set of *anchors* by

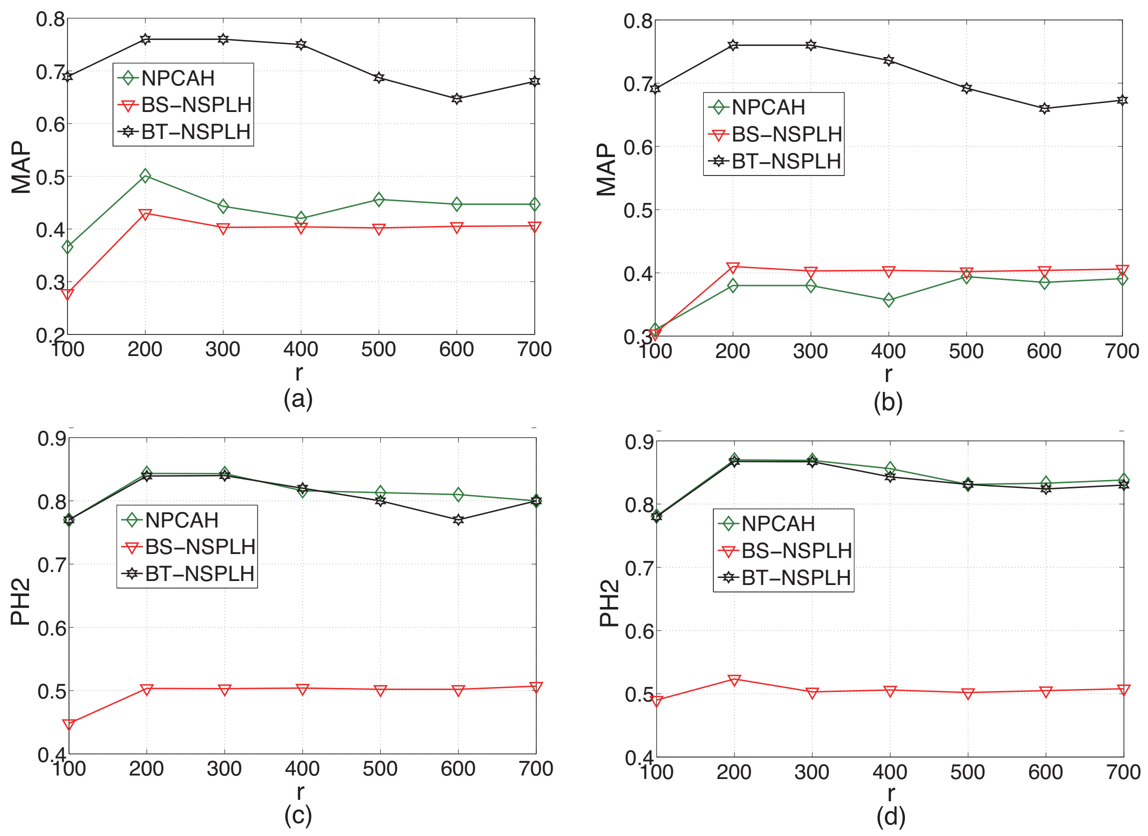


Fig. 2. Parameter selection results on MNIST. (a) MAP varies with the number of *anchors* r for 32-bit (b) MAP varies with the number of *anchors* r for 64-bit (c) PH2 varies with the number of *anchors* r for 32-bit (d) PH2 varies with the number of *anchors* r for 64-bit

running K-means on a small subset of training sets with 10 iterations. Other parameters[1] are carefully tuned for the best performance.

6.2 Evaluation Metric

Three hashing performance measure are adopted in our empirical evaluations: Mean Average Precision (MAP), Precision within Hamming radius 2 (PH2) and Recall.

Hamming ranking is employed to evaluate the MAP and Recall. For a given dataset, we issue the query with each point in the query set, and all points in the training set are ranked according to their Hamming distances from the query. Due to the binary representation, Hamming ranking is essentially fast in practice.

Average precision is defined as below:

$$\text{Average Precision} = \frac{1}{\sum_{i=1}^{n} v_i} \sum_{i=1}^{n} v_i \left(\frac{\sum_{j=1}^{i} v_j}{i} \right) \tag{24}$$

where v_i is set to one if the i-th point in rank list has the same label as the query, and n is the size of the entire dataset. MAP is the mean of average precision for all the queries in the query set, which approximates the area under precision-recall curve [29]. In our experiments, MAP is evaluated from 8-bit to 64-bit for all compared methods.

Recall is computed by:

$$\text{Recall} = \frac{\sum_{i=1}^{v_s} v_i}{\sum_{i=1}^{n} v_i} \tag{25}$$

where v_s is the size of scanned points in the rank list. Here the mean of Recall for all the queries in the query set is evaluated at 32-bit for all compared methods.

Hash lookup is used to evaluate the PH2, which emphasizes more on search speed since it has constant query time. When using many hash bits and a single hash table, however, hash lookup often fails because the Hamming space becomes increasingly sparse and very few samples fall in the same hash bucket. Hence, similar to [34], we search within a Hamming radius 2 to retrieve potential neighbors for each query. Precision within Hamming radius 2 is computed by:

$$\text{PH2} = \frac{\sum_{i=1}^{v_s} v_i}{v_s} \tag{26}$$

where v_i is set to one if the i-th returned point within Hamming radius 2 to the query has the same label as the query and v_s is the total number of returned points within Hamming radius 2 to the query. Here the mean of PH2 for all the queries in the query set is evaluated from 8-bit to 64-bit for all compared methods.

1. Our unoptimized Matlab implementation is publicly available from http://www.cse.cuhk.edu.hk/~jkzhu/bnsplh

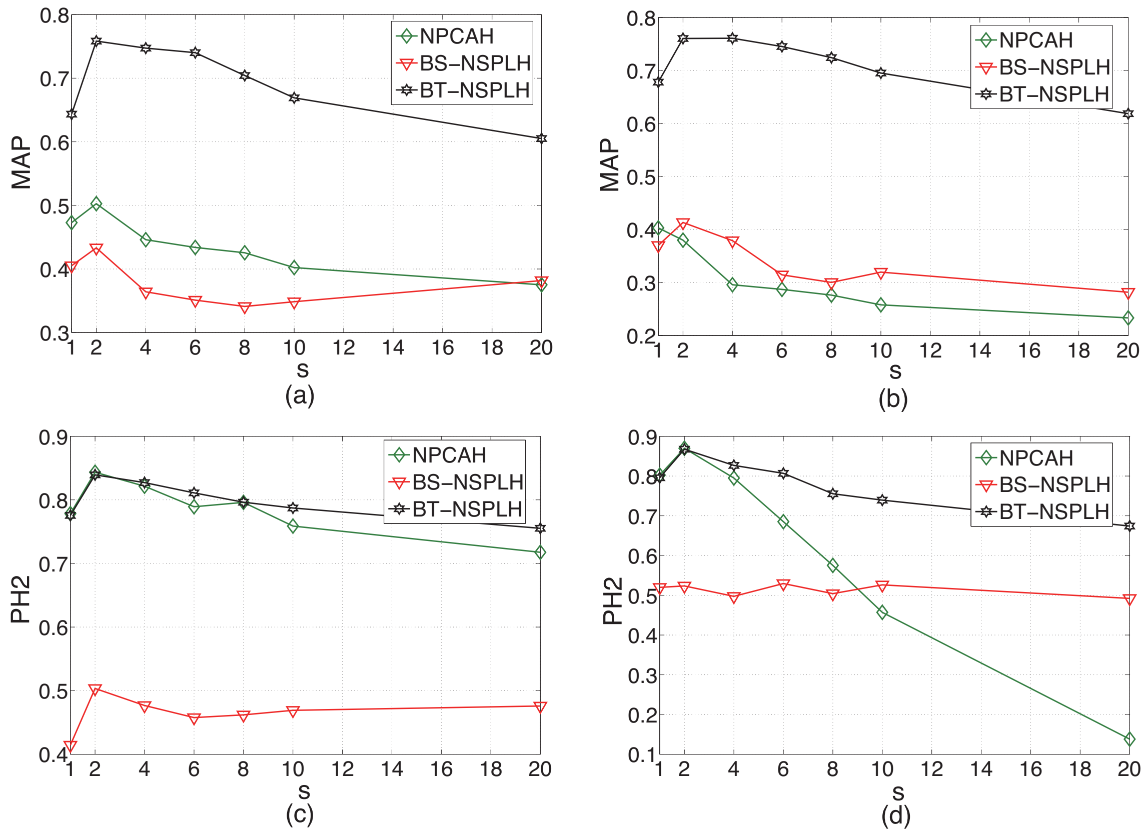


Fig. 3. Parameter selection results on MNIST. (a) MAP varies with the number of chosen nearest points s for 32-bit (b) MAP varies with the number of chosen nearest points s for 64-bit (c) PH2 varies with the number of chosen nearest points s for 32-bit (d) PH2 varies with the number of chosen nearest points s for 64-bit

6.3 Parameters Selection

In order to comprehensively evaluate our proposed approach, we choose the MNIST data set to conduct a thorough study on the main parameters used in the algorithm.

Firstly, we study how MAP and PH2 varying with the number of *anchors* r and the total number of chosen nearest points s for three nonlinear algorithms NPCAH, Boosting-NSPLH and Bootstrap-NSPLH. Other parameters keep unchanged during studying these two parameters. Figure 2 plots the experimental results varying with the number of *anchors* r from 100 to 700 for 32-bit and 64-bit when $s = 2$. We can find that three methods keep steady with r, and the proposed Bootstrap-NSPLH method achieves the best MAP performance. Moreover, Bootstrap-NSPLH and NPCAH perform much better than Boosting-NSPLH in PH2 performance. Figure 3 plots the results varying with the number of chosen nearest points s from 1 to 20 for 32-bit and 64-bit when $r = 300$. It can be seen that the proposed Bootstrap-NSPLH method outperforms the other two nonlinear methods at the large margin, though the performance decays slightly after $s = 2$. It can also be found that the performance of NPCAH decreases quickly when s increases especially for 64-bit since it does not consider HAE, which leads to the poor quality of high bits. In the following experiments, we choose $r = 300, s = 2$, as it obtains the best result for all three nonlinear methods.

Secondly, two key parameters of our proposed bootstrap learning scheme, α and β, are carefully studied. In our experiments, we find $\lambda = 8$ achieves the best performance while other parameters keep unchanged. Thus, λ is set to 8 and the number of *anchors* r is set to 300, the number of chosen nearest points s is set to 2 as discussed above. Figure 4 shows the MAP and PH2 varying with α and β for 32-bit and 64-bit. It is interesting to find that α which controls the positive pairs error slightly affects the performance while β which controls the negative pairs error should be carefully considered. This property is desirable since the negative pairs are much easier to obtain rather than the positive pairs in the real applications which means our proposed bootstrap learning scheme will perform well even with little precious positive pairs. Moreover, in Figure 4 one can easily find that a wide range of β, about from -0.8 to 0.2, can make desirable performance. Therefore, the proposed bootstrap learning scheme has a wide range to choose the two key parameters α and β.

6.4 Results

In the following experiments, we conduct the intensive evaluation on the seven hashing methods using six datasets.

Figure 5 plots the MAP results ranging from 8-bit to 64-bit for all methods. We can first observe that the

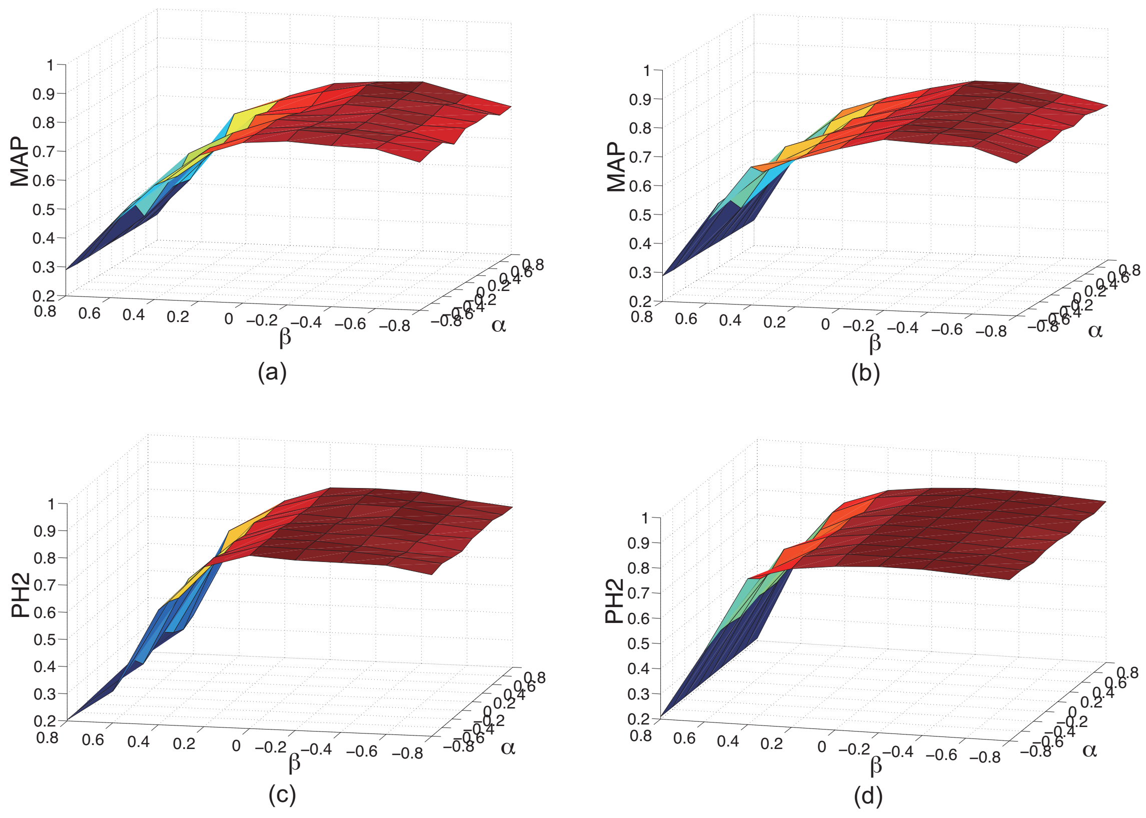


Fig. 4. Parameter selection results on MNIST. (a) MAP varies with parameters α and β for 32-bit (b) MAP varies with parameters α and β for 64-bit (c) PH2 varies with parameters α and β for 32-bit (d) PH2 varies with parameters α and β for 64-bit

nonlinear projection approach significantly improves the MAP performance for PCAH and the proposed Bootstrap-SPLH method on all the six datasets. This indicates that the nonlinear projection is able to capture the underlying relationship among the data points. Also, it is surprising to see that the boosting learning scheme performs even poorer with the nonlinear projection on most datasets, which may be due to its deficiency on the error correction. Moreover, we find that the MAP of PCAH and NPCAH drop quickly after 24-bit or 32-bit. This is mainly because the HAE is not considered during converting the real-value eigenvectors to the binary codes. Furthermore, it can be clearly seen that the proposed bootstrap sequential learning scheme outperforms the other methods significantly. Especially the Bootstrap-NSPLH shows a much higher MAP when the bit length increases.

TABLE 2
MAP results on MNIST for AGH-2 and Bootstrap-NSPLH

	AGH-2	Bootstrap-NSPLH
24bit	0.6738	0.7658
48bit	0.6410	0.7676

Additionally, we compare the presented Bootstrap-NSPLH method against AGH-2 algorithm [19], and the experimental results for 24-bit and 48-bit are shown in Table 2. We can find that the presented method outperforms AGH-2 at a large margin. We can also see that the performance for AGH-2 drops when bit length increases since it just slows down the effect incurred by HAE twice but not corrects it as we analyzed in Section1. In summary, the promising results indicate that the presented error correction scheme is very effective.

To qualitatively illustrate the hashing performance, we retrieve the top neighbors by the different techniques on one example digit. As shown in Figure 1, we find that Bootstrap-NSPLH tends to return more semantically consistent neighbors with 24-bit hashing codes.

Figure 6 shows the PH2 results ranging from 8-bit to 64-bit for all the algorithms. It can be seen that the precision for the hashing methods using linear projection drops after 24-bit or 32-bit. This is because the number of points falling in a bucket decrease exponentially with the increased sparsity of the Hamming space for more bits. Thus, many queries fail by not returning any neighbor even in a Hamming ball of radius 2. This shows a typical problem using hash lookup tables even though it is faster than Hamming ranking. Even in this case, Bootstrap-SPLH provides the best performance in the methods using linear projection. However, the methods using nonlinear projection do not suffer from this drawback. Both Bootstrap-NSPLH and NPCAH provide higher precision for more than 32-bit.

Figure 7 shows the recall curve in the case of 32-bit for each method. As the MAP results, bootstrap sequential learning scheme outperforms others significantly. Also, Bootstrap-NSPLH obtains the best performance.

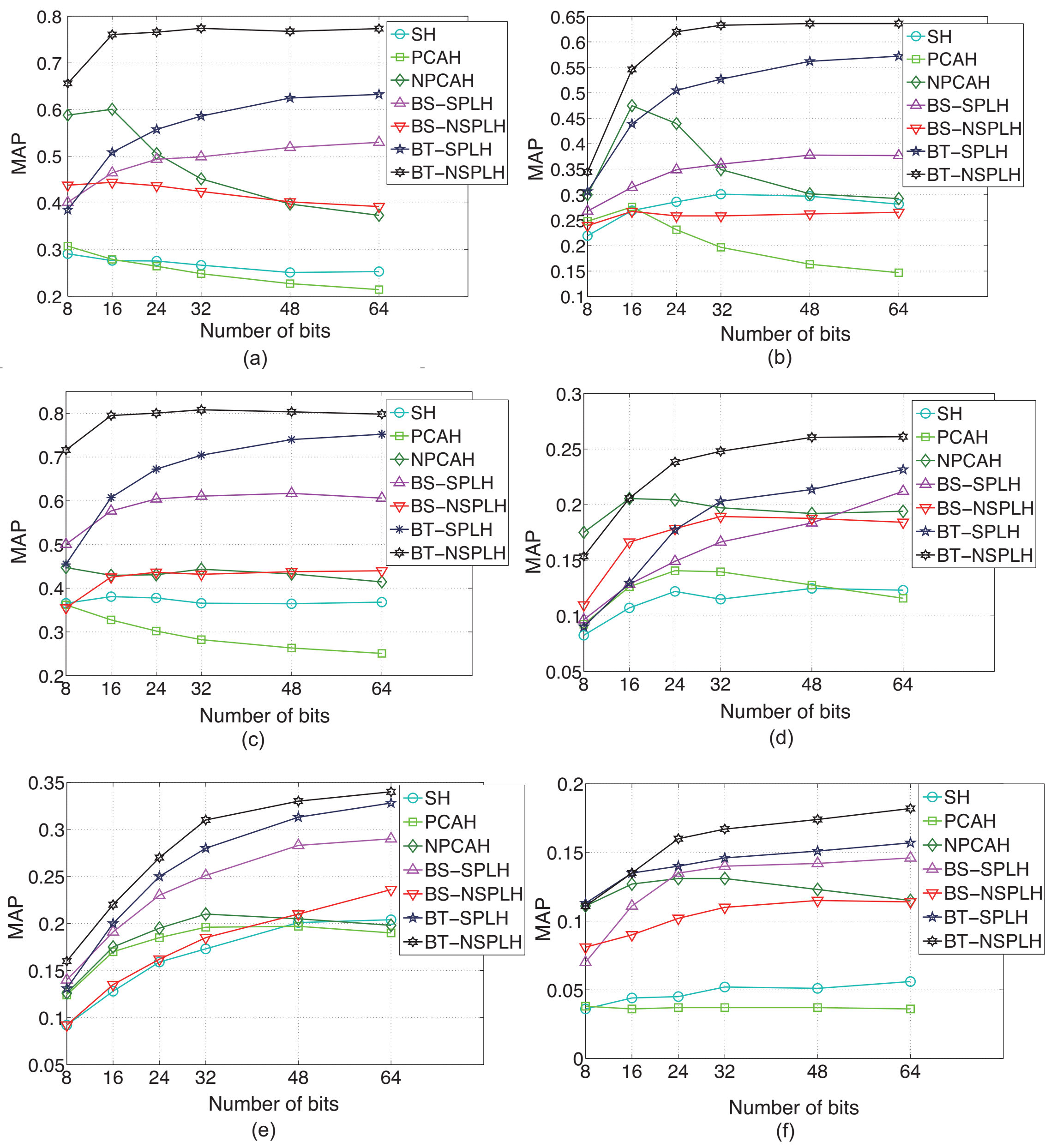


Fig. 5. MAP results: (a) MNIST (b) ISOLET (c) USPS (d) Caltech101 (e) SIFT1M (f) PATCH1M

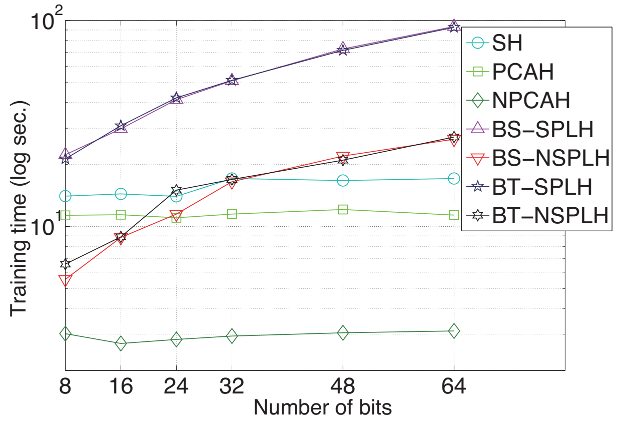


Fig. 8. Training time on MNIST

We empirically study the training time for each method on MNIST, and plot the result in Figure 8. It can be found that the sequential learning methods require more training time, which increases along with the bit length. We may notice that the time complexity for the presented bootstrap approach is almost the same as the boosting sequential learning method. Also, the methods using nonlinear projection require less training time. This is because the total number of selected *anchors* is set to 300, which is less than the original dimension 784 of data points on MNIST. Thus, we can conclude that the nonlinear projection method not only costs less training time for high dimension data, but also achieves much better hashing performance especially for the proposed bootstrap sequential learning scheme.

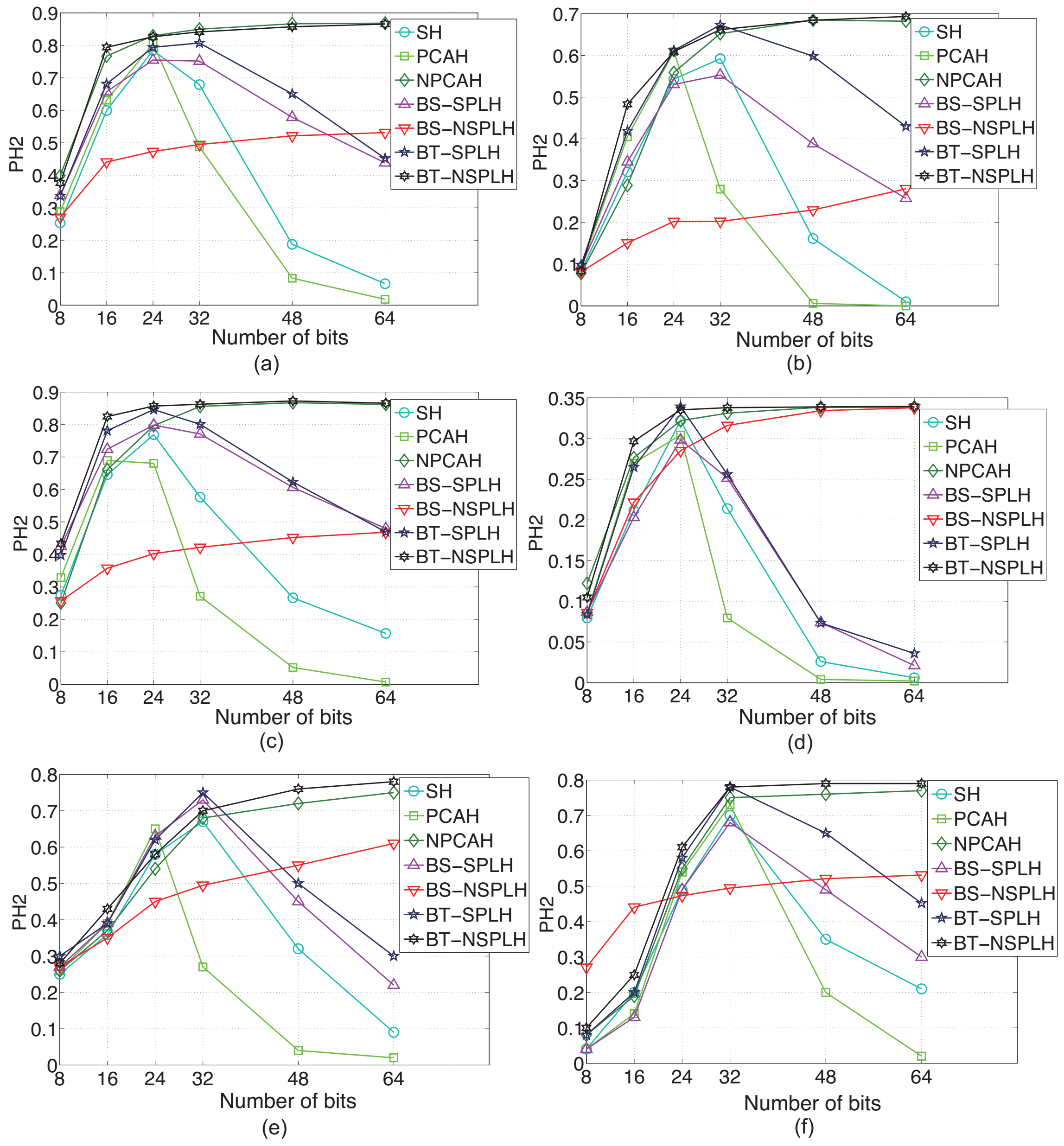


Fig. 6. PH2 results: (a) MNIST (b) ISOLET (c) USPS (d) Caltech101 (e) SIFT1M (f) PATCH1M

7 Conclusion and Future Work

In this paper, we present a Bootstrap Sequential Projection Learning method for semi-supervised nonlinear hashing, which firstly employs a nonlinear hash function to capture the underlying relationship among the data points for semi-supervised hashing. Secondly, the presented method takes advantage of a regularized learning-based objective function which can utilize both the labeled and unlabeled data. Finally, we apply a novel bootstrap-style sequential learning scheme to derive the hash function by correcting the errors in consideration of all previous learned bits holistically. The extensive experimental results show that the nonlinear hash function is much better than the linear one and the proposed bootstrap learning scheme achieves the best performance in contrast to other learning schemes.

In the future, we will extend our proposed scheme to the unsupervised case and apply it to some applications.

Acknowledgments

The authors appreciate the reviewers for their extensive and informative comments for the improvement of this manuscript. The authors would like to thank Mr. Jun Wang and his colleagues for providing their source code and datasets. The work was supported by National Natural Science Foundation of China under Grants (91120302 and 61103105), National Basic Research Program of China (973 Program) under Grant 2011CB302206, Fundamental Research Funds for the Central Universities, and Program for New Century Excellent Talents in University (NCET-09-0685).

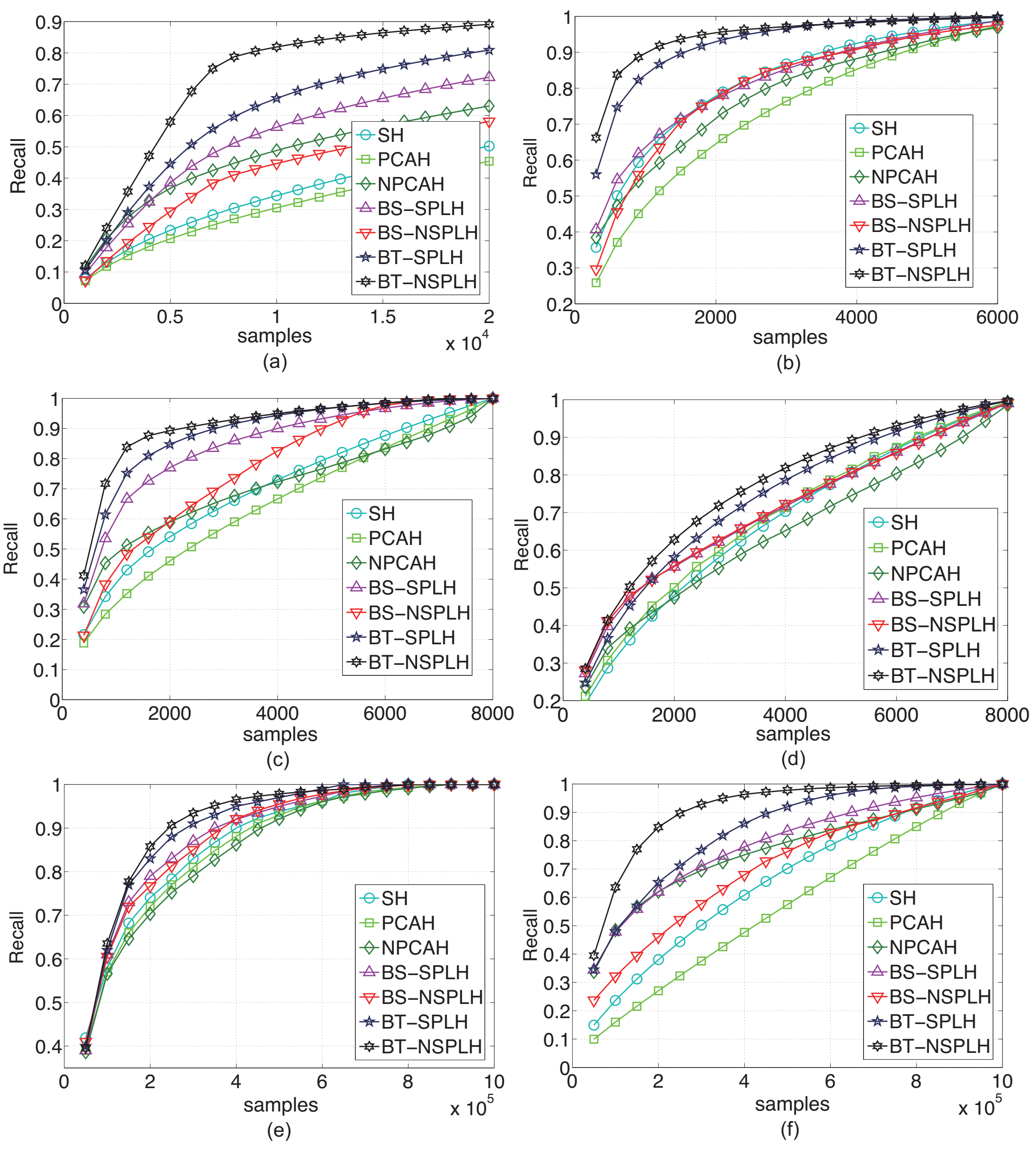


Fig. 7. (a) Recall curve for 32-bit: (a) MNIST (b) ISOLET (c) USPS (d) Caltech101 (e) SIFT1M (f) PATCH1M

References

[1] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*, 2006.

[2] A. M. Bronstein, M. M. Bronstein, L. J. Guibas, and M. Ovsjanikov. Shape google: Geometric words and expressions for invariant shape retrieval. *ACM Trans. Graph.*, 30:1:1–1:20, 2011.

[3] M. S. Charikar. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*, 2002.

[4] P. Ciaccia, M. Patella, and P. Zezula. M-tree: An efficient access method for similarity search in metric spaces. In *Proceedings of the 23rd International Conference on Very Large Data Bases*, 1997.

[5] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, 2004.

[6] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Comput. Vis. Image Underst.*, 106:59–70, 2007.

[7] J. H. Friedman, J. L. Bentley, and R. A. Finkel. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw.*, 3:209–226, 1977.

[8] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of the 25th International Conference on Very Large Data Bases*, 1999.

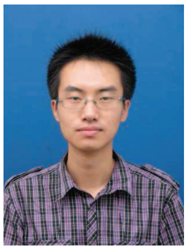
[9] K. Grauman and T. Darrell. Pyramid match hashing: Sub-linear time indexing over partial correspondences. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2007.

[10] J. He, W. Liu, and S.-F. Chang. Scalable similarity search with optimized kernel hashing. In *Proceedings of the 16th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2010.

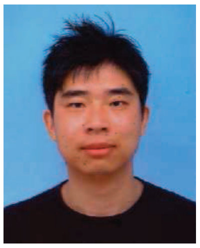
[11] M. Henzinger. Finding near-duplicate web pages: a large-scale evaluation of algorithms. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006.

[12] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth*

annual ACM symposium on Theory of computing, 1998.
[13] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, 2008.
[14] H. Jegou, M. Douze, and C. Schmid. Packing bag-of-features. In *IEEE 12th International Conference on Computer Vision*, 2009.
[15] Y. Koren. Factorization meets the neighborhood: a multifaceted collaborative filtering model. In *Proceeding of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, 2008.
[16] B. Kulis and T. Darrell. Learning to hash with binary reconstructive embeddings. In *Advances in Neural Information Processing Systems (NIPS)*, pages 1042–1050, 2009.
[17] M. S. Lew, N. Sebe, C. Djeraba, and R. Jain. Content-based multimedia information retrieval: State of the art and challenges. *ACM Trans. Multimedia Comput. Commun. Appl.*, 2:1–19, 2006.
[18] W. Liu, J. He, and S.-F. Chang. Large Graph Construction for Scalable Semi-Supervised Learning. In *Proceedings of the 27th International Conference on Machine Learning (ICML)*, 2010.
[19] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with Graphs. In *Proceedings of the 28th International Conference on Machine Learning (ICML)*, 2011.
[20] D. Lowe. Object recognition from local scale-invariant features. In *The Proceedings of the Seventh IEEE International Conference on Computer Vision*, 1999.
[21] M. Ozuysal, M. Calonder, V. Lepetit, and P. Fua. Fast keypoint recognition using random ferns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 32(3):448 –461, 2010.
[22] S. Pandey, A. Broder, F. Chierichetti, V. Josifovski, R. Kumar, and S. Vassilvitskii. Nearest-neighbor caching for content-match applications. In *Proceedings of the 18th international conference on World Wide Web*, 2009.
[23] R. Salakhutdinov and G. Hinton. Semantic hashing. *International Journal of Approximate Reasoning*, 50(7):969 – 978, 2009.
[24] C. Silpa-Anan and R. Hartley. Optimised kd-trees for fast image descriptor matching. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2008.
[25] B. Stein. Principles of hash-based text retrieval. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, 2007.
[26] B. Stein, S. M. zu Eissen, and M. Potthast. Strategies for retrieving plagiarized documents. In *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, 2007.
[27] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30:1958–1970, 2008.
[28] A. Torralba, R. Fergus, and Y. Weiss. Small codes and large image databases for recognition. 2008.
[29] A. Turpin and F. Scholer. User performance versus precision measures for simple search tasks. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, 2006.
[30] J. K. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, 1991.
[31] J. Wang, S. Kumar, and S.-F. Chang. Semi-supervised hashing for scalable image retrieval. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2010.
[32] 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 (ICML)*, 2010.
[33] J. Wang, J. Yang, K. Yu, F. Lv, T. Huang, and Y. Gong. Locality-constrained linear coding for image classification. 2010.
[34] Y. Weiss, A. B. Torralba, and R. Fergus. Spectral hashing. In *Advances in Neural Information Processing Systems (NIPS)*, 2008.
[35] Z. Yu, D. Cai, and X. He. Error-correcting output hashing in fast similarity search. In *The 2nd International Conference on Internet Multimedia Computing and Service*, 2010.
[36] D. Zhang, J. Wang, D. Cai, and J. Lu. Laplacian co-hashing of terms and documents. In *Proceedings of the 32nd European Conference on Information Retrieval*, 2010.
[37] D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. In *Proceeding of the 33rd international ACM SIGIR conference on Research and development in information retrieval*, 2010.



Chenxia Wu is currently a Master student in College of Computer Science at Zhejiang University. He received his BS degree in Computer Science from Southeast University, China. His research interests include machine learning, computer vision and multimedia information retrieval.



Jianke Zhu is an Associate Professor in College of Computer Science at Zhejiang University. He received his Ph.D degree in Computer Science and Engineering from Chinese University of Hong Kong. He was a postdoc in BIWI computer vision lab of ETH Zurich. Dr. Zhu's research interests include computer vision and multimedia information retrieval. He is a member of the IEEE.

Deng Cai is an Associate Professor in the State Key Lab of CAD&CG, College of Computer Science at Zhejiang University, China. He received the PhD degree in computer science from University of Illinois at Urbana Champaign in 2009. His research interests include machine learning, computer vision, data mining and information retrieval. He is a member of the IEEE.

Chun Chen received the PhD degree from Zhejiang University in 1990. He is a professor in College of Computer Science, the Dean of College of Software, and the Director of Institute of Computer Software at Zhejiang University. His research interests include image processing, computer vision, embedded system and information retrieval. He is a member of the IEEE.

Jiajun Bu received the BS and PhD degrees from Zhejiang University, in 1995 and 2000, respectively. He is a professor in College of Computer Science and the Deputy Director of China Research Center of Information and Accessibility Technology for Disability. His research interests include embedded system, information retrieval, data mining and information accessibility. He is a member of the IEEE.