Applied Intelligence (2022) 52:14724–14738
https://doi.org/10.1007/s10489-022-03546-9

Locality sensitive hashing with bit selection

Wenhua Zhou[1] · Huawen Liu[2] · Jungang Lou[3] · Xin Chen[1]

Accepted: 20 March 2022 / Published online: 31 May 2022

Abstract
Locality sensitive hashing (LSH), one of the most popular hashing techniques, has attracted considerable attention for nearest neighbor search in the field of image retrieval. It can achieve promising performance only if the number of the generated hash bits is large enough. However, more hash bits assembled to the binary codes contain massive redundant information and require more time cost and storage spaces. To alleviate this limitation, we propose a novel bit selection framework to pick important bits out of the hash bits generated by hashing techniques. Within the bit selection framework, we further exploit eleven evaluation criteria to measure the importance and similarity of each bit generated by LSH, so that the bits with high importance and less similarity are selected to assemble new binary codes. To demonstrate the effectiveness of the proposed framework of bit selection, we evaluated the proposed framework with the evaluation criteria on five commonly used data sets. Experimental results show the proposed bit selection framework works effectively in different cases, and the performance of LSH has not been degraded significantly after redundant hash bits reduced by the evaluation criteria.

Keywords Nearest neighbor search · Locality sensitive hashing · Hash bit · Bit selection · Binary code

1 Introduction

With the explosive growth of data such as images, documents and videos, nearest neighbor (NN) search (a.k.a similarity search) has attracted extensive attention in various domains including machine learning, information retrieval, natural language processing and outlier detection [1–3]. Given a query, the objective of NN search is to identify its nearest neighbors from a data collection. A variety of NN algorithms have been developed so far. However, finding exact nearest neighbors from large-scale data is prohibitive and infeasible from the perspective of efficiency [4].

In recent years, approximate NN (ANN) is becoming popular, for it finds approximate nearest neighbors within sub-linear and even constant query time [5]. Generally, the existing ANN search algorithms can be roughly divided into two categories, i.e, tree-based and hashing-based search algorithms. The tree-based search algorithms, such as KD-trees [6], M-trees [7], and ball-trees [8], exploit tree structures to store data and then find approximate nearest neighbors in sub-linear time [9]. However, the performance of these algorithms decreases dramatically when the dimension of data becomes high. Besides, they usually require large memory to store the tree structures.

The hashing-based search methods become flourishing recently because of their high query efficiency and low storage space [10]. They encode the high-dimensional data as binary-code representations by a series of hash functions. With the binary-code representations, retrieving nearest neighbors for a given query turns into matching the corresponding binary codes by bit operations. Benefiting from the bit operations, the task of ANN retrieval can be handled in main memory and extremely fast, usually within $O(1)$ time [11]. This enables the hashing-based search algorithms to be quite popular to large-scale scenarios.

This article belongs to the Topical Collection: *Special Issue on Multi-view Learning*
Guest Editors: Guoqing Chao, Xingquan Zhu, Weiping Ding, Jinbo Bi and Shiliang Sun

✉ Huawen Liu
huaw.liu@gmail.com

✉ Xin Chen
xinchen@zjnu.cn

[1] Department of Computer Science, Zhejiang Normal University, Jinhua 321004, People's Republic of China

[2] Department of Computer Science, Shaoxing University, Shaoxing 312000, People's Republic of China

[3] School of Information Engineering, Huzhou University, Huzhou 313000, People's Republic of China

Locality sensitive hashing (LSH), one of the most famous hashing algorithms, maps similar or proximate data samples into the same "bucket" with high probability [11]. Specifically, a series of hash functions are generated randomly to represent the similar data samples as binary codes with small Hamming distances, so that the neighborhood relations between the original data samples can be preserved. Since the hash functions within LSH are generated randomly, LSH has high efficiency. Owing to this, various extensions of LSH, including kernelized LSH (KLSH) [12], shift-invariant KLSH (SKLSH) [13]) and I-LSH [14], have been witnessed. It is noticeable that LSH can achieve desirable performance if the number of hash functions is large enough. This, however, leads to a high computation cost and storage overhead, limiting its applications to large-scale scenarios. For instance, the binary codes of one billion data samples with 1024 hash bits need around 1TB storage space, whereas the codes with 64 hash bits only need 64GB space.

On the other hand, data-dependent hashing techniques exploit inherent properties of data to learn elaborately hash functions, so that the generated binary codes have promising performance. Typical examples of such kind include principal components analysis hashing (PCAH) [15], iterative quantization (ITQ) [16], spectral hashing (SH) [17]. Anchor graph hashing (AGH) [18], sparse embedding and least variance encoding (SELVE) [19] and locally linear hashing (LLH) [20] utilize manifold structural information of data to derive the binary codes. Though the data-dependent hashing algorithms can generate powerful hash functions, their time complexity is relatively high and the scalability is not as good as one expects.

In this paper, we propose a novel bit selection framework to pick important bits out of the hash bits generated by hashing techniques. Though the hash functions are generated randomly, a large amount of redundant information exists within the binary codes assembled from the generated hash bits. It is necessary to remove those redundant information from the binary codes, making the assembled binary codes compact. Taking LSH as an example, we exploit eleven evaluation criteria to measure the importance and similarity of each hash bit generated by LSH, so that the bits with high importance and less similarity are selected to assemble new binary codes. The evaluation metrics not only consider individual bit's importance, e.g., Laplacian score (LS) [21] and information entropy (Ent) [22], but also involve correlations between a pairwise of hash bits, e.g., Pearson correlation coefficient (PCC) [23] and mutual information (MI) [24]. The experimental results conducted on public benchmark data sets show the proposed bit selection framework works effectively, and can achieve reduced effects of hash bits, without degrading the performance of LSH significantly.

The main contributions of this article are briefly highlighted as follows.

- A novel hash bit selection framework for hashing techniques is proposed. It aims to derive compact binary codes consisting of less hash bits, which embrace information as faithful as possible.
- Under the bit selection framework, we renders eleven bit selection methods by using different evaluation criteria to pick important and representative hash bits from the candidate hash bits generated by LSH, without degrading the performance significantly.
- Extensive experiments were carried out on public benchmark data sets. The experimental results show that the framework works effectively. The performance of LSH was not degraded significantly, after 20% - 60% redundant hash bits were removed.

The rest of this paper is organized as follows. Section 2 briefly reviews the state-of-the-art of hash learning methods. Section 3 introduces the bit selection framework for hashing techniques to derive compact binary codes. In Section 4, we propose eleven hash bit selection methods for LSH to show how the framework works. The Experimental results and discussions on public benchmark data sets are reported in Section 5, and the conclusions are presented in Section 6.

2 Related work

Due to their low storage cost and high computational efficiency, hashing techniques have been widely applied in a large variety of scenarios. As mentioned above, the hash learning algorithms can be roughly divided into two main groups: data-independent and data-dependent hashing algorithms [14]. This section presents a brief overview of the data-independent hashing algorithms. More details about the hashing techniques can be found in good surveys (e.g., [14]).

The data-independent hashing techniques have been extensively investigated to handle the problems of nearest neighbor search for large-scale data. Representative example of this kind is LSH, which maps similar data samples to proximate binary codes with high probability. Theoretically, the probability that two data samples have the same binary code is proportional to their similarity measured by Euclidean distance or semantic consistency [10, 11]. Since the hash functions of LSH are generated randomly, LSH have extremely high efficiency. With such favorable properties, a variety of LSH-like methods have been developed to learn informative and discriminative binary code. For instance, kernelized LSH (KLSH) [13] adopts arbitrary kernel functions to capture the intrinsic relationships among

data samples and can be employed in many existing image search measures. Shift-invariant kernelized LSH (SKLSH) [13] maps the random projections to shift-invariant kernel to maintain the similarity structures of the original data samples. Unfortunately, most of the LSH-like methods demand for a considerable length of the binary codes to ensure a competitive performance, leading to long computational time and high storage cost.

In contrast to the data-independent methods, the data-dependent hashing methods make full use of potential properties of data to construct more effective hash functions. Generally speaking, the data-dependent hashing methods can be divided into two classes, namely, principle component analysis (PCA) based hashing and manifold based hashing. The former attempts to preserve the maximal variance of hash bits in dimensionality reduction, during the generation process of hash bits. Unfortunately, maximizing the variance of hash bits is intractable because of the discrete constrain in learning binary codes. To deal with this issue, PCA hashing (PCAH) [15] switches to construct an set of orthogonal hash functions which are uncorrelated to each other, while iterative quantization (ITQ) [16], the most representative PCA-like hashing method, searches an orthogonal rotation matrix to maximizing the variance and minimizing the quantization error simultaneously.

Accordingly, the manifold-based hashing methods manage to make the generated binary codes preserve local similarity structures, a kind of nearest neighborhood relationships, among data pairs, wherein those data samples with similar properties possess the same binary values [19, 20]. Many of the manifold methods exploit various optimization structures to learn informative and important binary codes. For example, self-taught hashing (STH) [25] minimizes the weighted average Hamming distance formulation to meet the criterion of similarity preserving. Anchor graph hashing (AGH) [18] maintains the approximate neighborhood structures of the original data by introducing an anchor graph. Sparse hashing (SH) [26] converts the high-dimensional data samples to low-dimensional binary codes with a sparse coding technique, wherein the similarity structures are preserved with less storage.

Since the data-dependent hashing methods fully take the inherent information of data into consideration, they can learn more compact and have better performance in comparison to the LSH-like hashing methods. Nonetheless, their search efficiency is relatively low, where they need at least quadratic time during the training stage. Moreover, all of the data-dependent hashing methods focus on designing a set of effective hash functions, either maximizing the correlations or preserving the similarity structures, without taking into account the redundancy embraced within the generated binary codes.

Cai [11] argued that LSH is still competitive when the number of generated hash bits is large enough. However, the more the hash bits, the higher correlated between them. It is natural to reduce the number of generated hash bits. Liu et al. [27] considered the problem of bit selection by virtue of weighted graph and dynamic programming techniques. Specifically, for each hash bit, its quality and independence to others are represented as weighted vertex and weighted edges of a graph, respectively. Under this context, the problem of bit selection is formulated as quadratic programming on the graph, which can be solved by replicator dynamics. However, it is computationally expensive and less robust. Motivated by these observations, here we adopt eleven evaluation criteria to measure the importance and the similarities of the generated hash bits. Then the important and less similarity hash bits are picked out, and taken as representative ones to assemble new binary codes. With this strategy, the new binary codes can conserve the properties of the original data as faithful as possible.

3 Bit selection framework

Assume that $\mathbf{X} \in R^{n\times d}$ is a data set consisting of n data sample with d dimensions. For each row vector $\mathbf{x}_i$ of $\mathbf{X}$, it is the i-th d-dimensional data sample. Hash learning aims to implicitly or explicitly design a set of hash functions $\mathbf{H} = [\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_m] \in R^{d\times m}$ to encode the data samples $\mathbf{X}$ into binary representations $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m] \in \{-1, 1\}^{n\times m}$, where m is the number of hash bits and $m < d$. $\mathbf{b}_i \in \{-1, 1\}^n$ is the i-th hash bit, whose values are derived from the i-th hash function $\mathbf{h}_i$ on all data samples in $\mathbf{X}$.

Bit selection is the process of extracting l principal and representative hash bits from the m candidate hash bits generated by hashing techniques to take the place of the original ones when assembling binary codes. Formally, let $\mathbf{B} = [\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_m] \in \{-1, 1\}^{n\times m}$ be the binary codes assembled from the m hash bits. Since bit selection picks l hash bits out from the m bits without rectifying hash values, the new binary codes $\mathbf{B}' = [\mathbf{b}_1, \mathbf{b}_2, ..., \mathbf{b}_l] \in \{-1, 1\}^{n\times l}$ assembled from the l selected hash bits actually is a subset of $\mathbf{B}$, i.e., $\mathbf{B}' \subseteq \mathbf{B}$.

For this purpose, We propose a bit selection framework for hash learning. The selection framework is shown as Fig. 1. It comprises three major stages, namely, bit generation, bit selection and code assembly. Code assembly refers to the process of concatenating the l selected hash bits in a sequential way. Since it is relatively intuitive and simple, we place more focuses on discussing the first two steps.

The first step of our framework is to generate m candidate hash bits by using the off-the-shelf hashing techniques. For the hashing techniques used within, they can be only one

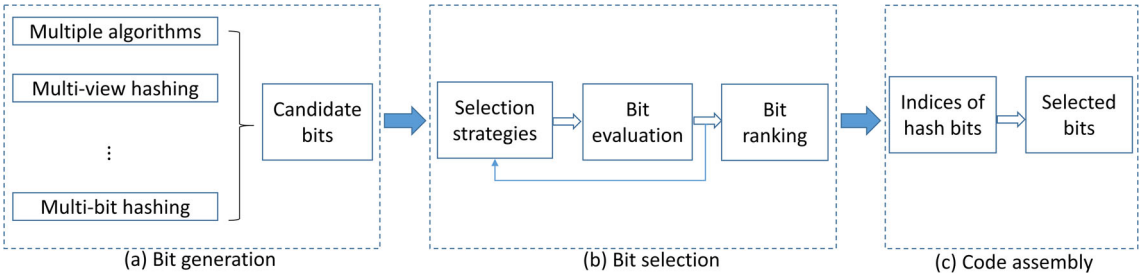

Fig. 1 The framework of hash bit selection for hash learning

type of hashing algorithm, or multiple hashing algorithms simultaneously. Even more, different types, e.g., the data-independent hashing and the data-dependent hashing, of hashing techniques can also be employed at one time. It should be mentioned that several hashing algorithms, such as PCAH, SH and ITQ, generate less hash bits. In this case, multiple hashing algorithms are often combined to generate a given number of candidate hash bits. For instance, if 600 hash bits are required to be under consideration, we can adopt LSH, PCAH and ITQ to generate 200 hash bits, respectively. Afterwards, these hash bits are concatenated to derive 600 hash bits.

Note that constructing the pool of candidate hash bits by multiple hashing algorithms is inherently consistent with multi-view learning, where data are collected from multiple sources [28]. Inspired by multi-view learning, we can also exploit multi-view hashing algorithms to construct the pool of candidate hash bits [29, 30]. Kong et al. [31] stated that a single hash bit per projection may separate similar samples into different binary values. Hence, multi-bit hashing algorithms encode each projection dimensional with multiple bits of binary values to map those close samples into same binary codes.

The second stage is bit selection, which picks representative and important hash bits out from the candidate bits according to evaluation criteria. The selected hash bits should embrace faithful information to the original one and have strong representation capability, so that new binary codes assembled from the selected bits have good performance.

Within this stage, three major components are involved and tightly associated to each other. They are selection strategies, bit evaluation and bit ranking. Select strategies denote which way of picking bits from the candidate bits is adopted; that is, choosing a bit individually or a subset of bits. If we treat bit individually, only one hash bit is evaluated and chosen at each time, and the most important bits are eventually chosen, according to evaluation criteria. On the contrary, multiple bits are evaluated as a whole and the subset of bits with high evaluation score is chosen to assemble binary codes.

Bit evaluation is a core component of bit selection, for it determines the importance or information amount for each hash bit or bit subset. To evaluate hash bits, evaluation criteria or metrics are required, and different criteria should be exploited for selection strategies. For instance, if a hash bit is under consideration, entropy or Laplacian score can be used to measure the importance of the hash bit; for a subset of hash bits, Pearson correlation coefficient and mutual information are good criteria to measure the correlation or similarity between pairwise bits.

Based on the important scores induced at the step of bit evaluation, bit ranking sorts the candidate hash bits or subsets in a descending order. The hash bits or subset with high important scores are chosen with high priority to assemble new binary codes.

4 Bit selection for LSH

LSH, one of the most popular hashing algorithms, attempts to map similar or proximate data samples to binary codes by using random hash functions. Let $\mathbf{h}$ be a randomly generated hash function, and $\mathbf{x} \in R^d$ be a data sample, the hash value of $\mathbf{x}$ is

$$b = sign(\mathbf{x}^T\mathbf{h}) = \begin{cases} 1, & if \quad \mathbf{x}^T\mathbf{h} \geq 0; \\ -1, & otherwise. \end{cases} \tag{1}$$

Given m random hash functions $\mathbf{H} = \{\mathbf{h}_1, \mathbf{h}_2, ..., \mathbf{h}_m\}$ and data samples $\mathbf{X} \in R^{n \times d}$, the i-th hash bit of $\mathbf{X}$ is

$$\mathbf{b}_i = sign(\mathbf{X}\mathbf{h}_i) \tag{2}$$

With $\mathbf{H}$, the k-th sample $\mathbf{x}_k$ is represented as the following binary code

$$\mathbf{c}_k = sign(\mathbf{x}_k^T\mathbf{H}) = [\mathbf{b}_{k1}, \mathbf{b}_{k2}, .., \mathbf{b}_{km}] \tag{3}$$

Theoretically, there is a high probability that two similar or proximate data samples are represented as the same binary code. Unfortunately, a great number of hash functions are required to retain this feature, that is, the binary codes should be long enough so that LSH is able to deliver a remarkable performance. Long binary code

generation demand on high computational cost and large storage space, and the generated hash bits will contain an increasing body of redundancy as its length increases.

A possible solution to the bit redundancy is performing bit selection for LSH. Under the above selection framework of hash bits, here we render eleven bit selection methods to pick important hash bits to assemble new binary codes. To be specific, a pool of candidate hash bits is first constructed from the generated hash bits by virtue of LSH. Then each candidate bit is evaluated in terms of selection criteria. Afterwards, the candidate bits are sorted according to the evaluation scores, and only those candidate bits with high scores are used to assemble new binary codes.

Two types of evaluation criteria are adopted in our selection methods to measure the importance or similarity for each candidate hash bit: one evaluates candidate bit individually, and the other evaluates multiple candidate bits as a whole.

4.1 Bit importance

The bit importance criteria assess an individual bit by calculating several metrics including balance, variance, entrophy, similarity preservation, and Laplacian score.

1. Variance (Var). In probability theory, the variance of a random variable measures the variation degree of its values from the mathematical expectation. A large variance indicates the values differ far from each other. The mathematical definition of variance of $\mathbf{b}_i$ is shown as follows.

$$Var(\mathbf{b}_i) = \frac{\sum_{k=1}^{n}(b_{ik} - \overline{b_i})^2}{n-1} \tag{4}$$

   where b_{ik} is the k-th binary value of $\mathbf{b}_i$, and $\overline{b_i}$ is the mean value of $\mathbf{b}_i$. From the definition, the larger the variable, the more information the hash bit carries.
2. Balance (Bal) [32]. Given a hash bit $\mathbf{b}_i$, the balance of $\mathbf{b}_i$ refers to the equilibrium degree of 1 to -1 within $\mathbf{b}_i$. From the perspective of data, the balance implies the representation capability of the corresponding hash function $\mathbf{h}_i$, which encodes the data samples to one of binary values, -1 and 1. Formally, the balance of the i-th hash bit $\mathbf{b}_i$ is $Bal(\mathbf{b}_i) = \mathbf{b}_i^T\mathbf{1}$. It is remarked that a hash bit with rich information and powerful discernibility can partition the data samples evenly [32], so a smaller $Bal(\mathbf{b}_i)$ indicates the better balance of $\mathbf{b}_i$. For the sake of consistency, here the balance of $\mathbf{b}_i$ is defined as

$$Bal(\mathbf{b}_i) = 1 - \frac{|\sum_{k=1}^{n} b_{ik}|}{n} \tag{5}$$

3. Entropy (Ent) [22]. Entropy is an effective criterion to measure the uncertainty degree a random variable has in information theory. Given a variable, it contains much more information and more important to prediction if it is highly uncertain. For the hash bit $\mathbf{b}_i$, its entropy is represented as

$$Ent(\mathbf{b}_i) = -\sum_{v\in\{1,-1\}}\sum_{k} P(b_{ik}=v)\log P(b_{ik}=v) \tag{6}$$

   where b_{ik} is the k-th value of $\mathbf{b}_i$ and $P(b_{ik} = v)$ is probability of b_{ik}=v (v=1 or -1).
4. Similarity preservation (SP) [27]. Similarity preservation denotes the similarities of data in the original space should be preserved after projected into the Hamming space; that is, the structural property of data should also be kept down, and similar data samples should be presented as similar binary codes. Thus, a high-quality hash bit can preserve the local structural information of data. Specifically, the metric of similarity preservation of the i-th bit is

$$SP(\mathbf{b}_i) = \left\|\mathbf{b}_i\mathbf{b}_i^T - \mathbf{S}\right\|^2 \tag{7}$$

   where $\mathbf{S}$ is an affinity matrix and each entry $S_{i,j} = \exp(-\left\|\mathbf{x}_i - \mathbf{x}_j\right\|^2/\epsilon^2)$. ϵ is a constant parameter. Note that, the smaller the value of $SP(\mathbf{b}_i)$, the higher the quality of the i-th hash bit.
5. Laplacian Score (LS) [21]. Laplacian score is widely used to evaluate quality of variable. It considers both local similarity property and variance of data simultaneously. Given a hash bit $\mathbf{b}_i$, its Laplacian score is represented as follows.

$$LS(\mathbf{b}_i) = -\frac{\sum_{k,j}(b_{ik} - b_{ij})^2 S_{kj}}{Var(\mathbf{b}_i)} \tag{8}$$

   where b_{ik} is the k-th value of $\mathbf{b}_i$, and S_{kj} is the similarity degree of $\mathbf{x}_k$ to $\mathbf{x}_j$. A high Laplacian score of $\mathbf{b}_i$ denotes the hash bit is important, for it can retain the local similarities of data and has a small variance simultaneously.

According to the definitions above, we know that these metrics evaluate the importance of hash bits individually. With these evaluation metrics, the principle of bit selection is to reserve those hash bits with high scores; that is, if a bit has high score, it will be selected to assemble new binary codes with a high probability.

Algorithm 1 summarizes the implementation details of our bit selection methods by evaluating individually hash bits. It works in a straightforward way and can be easily understood. Firstly, LSH is performed to generate m candidate hash bits, forming a pool of bits. For each candidate bit, its importance is evaluated by virtue of one of the corresponding metrics ((4) to (8)). Afterwards, the candidate hash bits are sorted in a descending order in terms of

their importance. Finally, the top l hash bits are kept down to assemble new binary codes.

Algorithm 1 Bit selection with individual bit evaluation.

Input: The data set $\mathbf{X} \in R^{n \times d}$, the number of candidate bits m, and the number of the selected bit l;

Output: The binary codes with the selected bits $\mathbf{B}' \in \{-1, 1\}^{n \times l}$;

1: Generate m candidate hash bits $\mathbf{B} \in \{-1, 1\}^{n \times m}$ by LSH;
2: **For** i=1 to m **do**
3: Calculate the importance of $\mathbf{b}_i$ by (4)-(8);
4: **End**
5: Rank the importance of the candidate bits in descending order;
6: Select the top l bits to assemble new binary codes $\mathbf{B}'$;
7: Return $\mathbf{B}'$.

The computational cost of Algorithm 1 mainly consists of three steps: generating the candidate bits (Step 1), estimating the important scores (from Step 2 to 4) and ranking the scores (Step 5). The first step conduct LSH to generate the candidate bits $\mathbf{B}$. Its computational time is $\mathcal{O}(mnd)$, where m is the number of hash bits. For the second step, its time complexity is dependent on which evaluation criterion is adopted. Generally, the complexity is a linear one, i.e., $\mathcal{O}(km)$, where k is a constant value. The computational time of bit ranking is $\mathcal{O}(m \log m)$. Hence, the total time cost of Algorithm 1 is $\mathcal{O}(mnd + m \log m)$.

4.2 Bit similarity

It is worth to mention that the metrics above only take the importance of individual bit into consideration, ignoring the interactions between bits. This may unfortunately leads to a fact that the selected hash bits contain redundant information. It is notable that highly correlated bits often exhibit similar behaviors. For example, two highly correlated hash bits $\mathbf{b}_i$ and $\mathbf{b}_j$ assign the same value, i.e. 1 or -1, to any sample, that is the two bits contribute equally to the hashing learning. In such a case, redundancy exists between $\mathbf{b}_i$ and $\mathbf{b}_j$, and it is apparently no need to choose both $\mathbf{b}_i$ and $\mathbf{b}_j$ for compact code assembling even though they both present high importance. As far as the bit redundancy is concerned, we take six more metrics to evaluate the pairwise similarities among hash bits. Below is the brief description of these metrics.

1. Hamming Distance (HD). Hamming distance is frequently used to measure how many different values between two strings at the same positions. Here we exploit it to check how many different hash values between two given hash bits. Given two hash bits $\mathbf{b}_i$ and $\mathbf{b}_j$, their Hamming distance is

$$HD(\mathbf{b}_i, \mathbf{b}_j) = \exp\left(-\sum_k b_{ik} \oplus b_{jk}\right) \tag{9}$$

where $\oplus$ refers to the logical operation of exclusive disjunction that returns 1 only when b_{ik} is differ to b_{jk}; otherwise returns 0. According to the definition, one may observe that if $\mathbf{b}_i$ has a small Hamming distance to $\mathbf{b}_j$, they are proximate to each other; that is, they are highly correlated.

2. Euclidean Distance (ED). Traditionally, the distance refers to how far from one sample to another in the Cartesian coordinate. Here we just calculate the distance between two hash bits with binary values. To be specific, the Euclidean distance $ED(\mathbf{b}_i, \mathbf{b}_j)$ of $\mathbf{b}_i$ to $\mathbf{b}_j$ is shown in the following.

$$ED(\mathbf{b}_i, \mathbf{b}_j) = \exp\left(-\sum_{k=1}^{n} (b_{ik} - b_{ij})^2\right) \tag{10}$$

where b_{ik} is the k-th value of $\mathbf{b}_i$. Obviously, the smaller the Euclidean distance $ED(\mathbf{b}_i, \mathbf{b}_j)$, the more similar between $\mathbf{b}_i$ and $\mathbf{b}_j$.

3. Cosine Distance (CD). Generally, this measurement denotes a cosine value of the angle between two random variables. If these two variables have the close orientation, the distance is very small. Particularly, the cosine similarity equals to 1 if the angle is zero. On the contrary, the similarity is zero if these two variable are perpendicular to each other. Hence, we use the cosine distance to approximate the similarity of $\mathbf{b}_i$ to $\mathbf{b}_j$ as follows

$$CD(\mathbf{b}_i, \mathbf{b}_j) = \cos(\mathbf{b}_i, \mathbf{b}_j) = \frac{\mathbf{b}_i^T \mathbf{b}_j}{\|\mathbf{b}_i\| \left\|\mathbf{b}_j\right\|} \tag{11}$$

4. Jaccard Distance (JD). The Jaccard distance (a.k.a. Jaccard similarity coefficient) is often taken to represent the similarity between two sets. It refers to the proportion of how many values are the same within the sets to the number of all values within their union set. Within each hash bit, its hash value is a binary one. Thus, the union and intersection sets are not appropriate. For the purpose of bit evaluation, the Jaccard distance of $\mathbf{b}_i$ to $\mathbf{b}_j$ is given by

$$JD(\mathbf{b}_i, \mathbf{b}_j) = \frac{\sum_k I(b_{ik} = b_{jk})}{\sum_k I(b_{ik} = 0) + \sum_k I(b_{jk} = 0) - \sum_k I(b_{ik} = b_{jk})} \tag{12}$$

where $I(v)$ is an indication function of v. If v is true, $I(v) = 1$; otherwise, $I(v) = 0$.

5. Pearson Correlation Coefficient (PCC) [23]. Pearson correlation coefficient is a popular criterion to measure the correlation between two variables in statistics. For the hash bits $\mathbf{b}_i$ and $\mathbf{b}_j$, their correlation coefficient is

$$PCC(\mathbf{b}_i, \mathbf{b}_j) = \frac{Cov(\mathbf{b}_i, \mathbf{b}_j)}{\sqrt{Var(\mathbf{b}_i)Var(\mathbf{b}_j)}} \quad (13)$$

where $Cov(\mathbf{b}_i, \mathbf{b}_j)$ is the covariance of $\mathbf{b}_i$ to $\mathbf{b}_j$, and $Var(\mathbf{b}_i)$ is the variance of $\mathbf{b}_i$. A large value of $PCC(\mathbf{b}_i, \mathbf{b}_j)$ implies that $\mathbf{b}_i$ and $\mathbf{b}_j$ are highly correlated to each other, making them more similar.

6. Mutual Information (MI) [24]. Mutual information can effectively measure how much information commonly shared by two variables. If one variable contains a large amount of information through observing another one, they may exhibit similar properties or behaviors. Based on the notion of entropy, the mutual information between $\mathbf{b}_i$ and $\mathbf{b}_j$ is defined as follows.

$$MI(\mathbf{b}_i, \mathbf{b}_j) = Ent(\mathbf{b}_i) + Ent(\mathbf{b}_j) - Ent(\mathbf{b}_i, \mathbf{b}_j) \quad (14)$$

where $Ent(\mathbf{b}_i)$ is the entropy of $\mathbf{b}_i$ in (6).

Based on the similarity metrics above, we render the second kind of bit selection methods for LSH in Algorithm 2. Contrastive to the selection methods in Algorithm 1 which only involves the importance of individual bit, Algorithm 2 mainly take use of the similarities of pairwise bits to choose those important bits with less correlation. To be specific, we firstly employ LSH to yield a pool of candidate hash bits. The similarity matrix $\mathbf{A}$ of the hash bits is derived by calculating the similarities between bits in a pairwise way using the metrics provided above ((9) to (14)). To reduce the redundancy among the selected hash bits, we choose greedily the most important bits during the iteration process of bit selection, that is, once the maximum value a_{ij} in $\mathbf{A}$ is picked, the i-th or j-th bit exclusively, and remove the i-th and j-th rows and columns from $\mathbf{A}$. The selection process is repeated until l bits are obtained, which are subsequently used to assemble more compact binary codes.

The computational cost of Algorithm 2 mainly comprises three steps: generating the candidate bits (Step 1), estimating the similarity scores between pairwise bits (from Step 2 to 6) and selecting the first l hash bits (from Step 8 to 12). The time cost of the first step is $\mathcal{O}(mnd)$. Since the bit similarity is estimated via a pairwise way, its computational time is $\mathcal{O}(m^2)$. Within the last step, a column and a row are removed at each time. Thus, its time complexity is $\mathcal{O}(m)$. Hence, the total time cost of Algorithm 2 is $\mathcal{O}(mnd + m^2 + ml)$.

Algorithm 2 Bit selection with bit correlation.

Input: The data set $\mathbf{X} \in R^{n \times d}$, the number of candidate bits m, and the number of the selected bit l;

Output: The binary codes with the selected bits $\mathbf{B}' \in \{-1, 1\}^{n \times l}$;

1: Obtain m candidate hash bits $\mathbf{B} \in \{-1, 1\}^{n \times m}$ by LSH;
2: **For** i=1 to m **do**
3: **For** j=1 to m **do**
4: Calculate the correlation a_{ij} between $\mathbf{b}_i$ and $\mathbf{b}_j$ by (9)-(14);
5: **End**
6: **End**
7: Construct the similarity matrix $\mathbf{A}=[a_{ij}]_{i,j=1..m}$;
8: **For** i=1 to l **do**
9: Locate the row-index of the largest value a_{ij};
10: Select the i-th or j-th hash bit exclusively;
11: Update $\mathbf{A}$ by removing the i-th and j-th columns and rows;
12: **End**
13: Assemble new binary codes $\mathbf{B}'$ by using the rest bits in $\mathbf{A}$;
14: Return $\mathbf{B}'$.

5 Evaluation experiments

To verify the effectiveness of the proposed framework of bit selection, we conducted a series of experiments on public data sets. Specifically, we made a comparison of LSH to its corresponding ones with the selection criteria. Besides, NDomSet [27], a kind of bit selection algorithm developed recently, was also used as a baseline.

The objectives of evaluation experiments are three-fold; that is, whether the performance of LSH is degraded significantly after bit selection performed, how the performance of LSH is changed with different numbers of selected hash bits, how many the selected bits are required without degraded the performance of LSH significantly. For these purposes, we carried out three groups of experiments on a PC with a 3.8GHz Intel Core i7-9700 CPU and 8GB RAM.

5.1 Experimental data sets

Five publicly available image data sets were employed in our experiments. They are CALTECH101, CIFAR-10, USPS, IAPR-TC12 and NUS-WIDE. Their brief descriptions are given in the following.

- CALTECH101 consists of 8,677 color images with 101 categories, about 50 images per category [33]. The image resolution is roughly 300×200 pixels, and each image is represented as a 512-dimensional GIST feature vector.
- CIFAR-10 contains 60,000 color images from the well known 80M tiny image collection [34]. These images are categorized into 10 classes, each consisting of 6,000 impages with 32×32 sizes. The images are also represented as 512-dimensional GIST feature vectors.
- USPS consists of 9,298 images (16×16) associated with a digit form '0' to '9' [20]. Each image is represented as a 256-dimensional feature vector.
- IAPR-TC12 is a multi-label data set that contains 20,000 image-text pairs annotated by 255 labels [35], covering natural images taken from locations around the world. Each image is represented by a 512-dimensional GIST feature vector.
- NUS-WIDE comprises of 269,648 images, where each image is annotated by 81 labels. In our experiments, the first 10 categorizes are considered, amounting to 186,577 images [17]. The ground-truth images are those that share at least one common label.

5.2 Evaluation metrics

Following the routine, in our experiments we also adopted three commonly used evaluation metrics, i.e., top-k precision (Pre@k), top-k recall (Rec@k) and mean average precision (mAP), to evaluate the performance of LSH. For each query sample, its retrieved nearest neighbors were ranked according to the Hamming distance, so that the evaluation criteria could be estimated. The evaluation criteria are defined as follows.

$$Pre@k = \frac{\text{the relevant images in the top-}k\text{ images}}{\text{the retrieved top-}k\text{ images}} \tag{15}$$

$$Rec@k = \frac{\text{the relevant images in the top-}k\text{ images}}{\text{all the relevant images}} \tag{16}$$

$$mAP = \frac{1}{Q}\sum_{i=1}^{Q}\frac{1}{M}\sum_{j=1}^{R}P_i(j)I_i(j) \tag{17}$$

where Q is the number of query samples, M is the number of relevant images, R is the number of retrieved images. $P_i(j)$ is the top-j precision of the i-th samples, and $I_i(j) = 1$ indicates the j-th retrieved image is a true neighbor of the i-th query, otherwise $I_i(j) = 0$. Without loss of generality, we retrieved 1,000 nearest samples for each query, i.e., $k = 1000$, in our experiments.

5.3 Results and discussion

5.3.1 Comparison of LSH to the bit selection methods

The first group of comparison experiments aims to determine how much the bit selection methods bring negative effects to the performance of LSH. For this purpose, we first performed LSH on the data sets to generate 128, 256 and 512 candidate hash bits, respectively. Then the bit selection methods were adopted to pick 80% bits out from the candidate bits; that is, 103, 205, 410 hash bits were chosen, respectively. Based on the selected hash bits, the values of the evaluation criteria of LSH and its corresponding ones with the selection methods were recorded.

The recall comparisons of LSH to the bit selection methods with 80% hash bits are presented in Fig. 2 to 6, where the bars above (or below) zero indicate that the bit selection methods have higher (or lower) values of the rec@k criterion than that of LSH. For example, the bit selection method using *Euclidean distance* (ED) achieved higher rec@k than LSH when the number of hash bits was 102, while the one using *entropy* (Ent) had worse performance than LSH on CALTECH101 (see Fig. 2).

From the experimental results in Figs. 2, 3, 4, 5 and 6, one can observe that the bit selection methods work well and have not deteriorated the performance of LSH significantly, after removing 20% redundant hash bits generated by LSH. In some cases, several selection methods even outperformed LSH. For instance, on the CALTECH101 data set, the selection methods using *Euclidean distance* (ED), *Jaccard distance* (JD) and *mutual information* (MI) achieved better performance with 80% hash bits, in comparing to LSH. Another interesting thing is that on all data sets, the difference of rec@k between the selection methods with 410 bits and LSH with 512 bits is smaller than those with less bits. This is intuitively reasonable because the more the hash bits, the more the redundancies among them.

The experimental comparisons of LSH with the bit selection methods with 80% hash bits at the aspect of precision are presented in Figs. 7, 8, 9, 10 and 11. According to the experimental results, similar conclusions can be easily made for the precision criterion. As an example, the bit selection methods with less hash bits on the CALTECH101 data set (see Fig. 7) are consistent with LSH in terms of precision; that is, the selection methods have not degraded the performance of LSH after removing 20% hash bits. On the CIFAR-10 data set (see Fig. 8 (a)-(c)), the difference of the selection methods with 410 hash bits are much smaller than those with 103 bits. This indicates the fact that long hash codes embody much more information, and

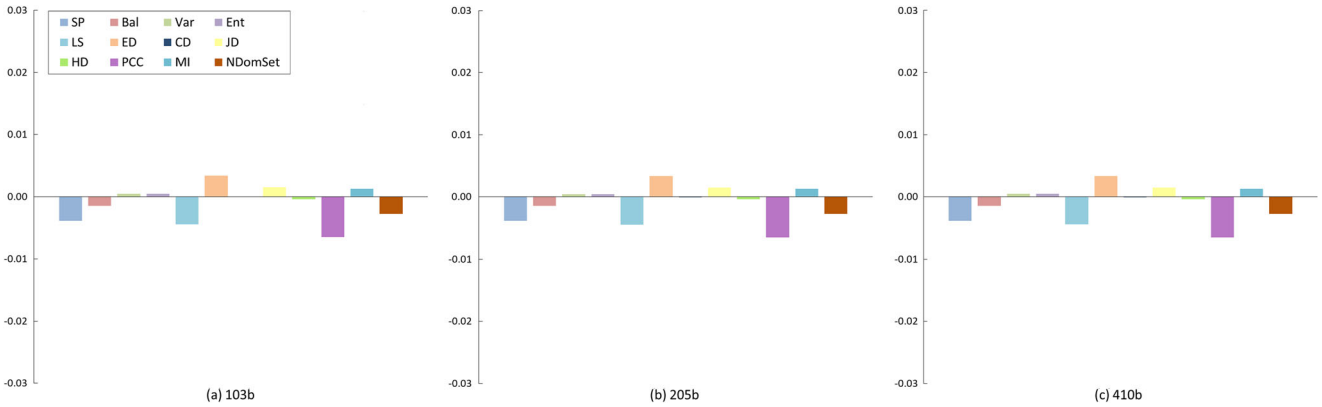

Fig. 2 The recall comparison of LSH to the bit selection methods with 80% hash bits on CALTECH101, where the bars above (below) zero indicate that they are better (worse) than LSH. (a) 103 over 128 bits (b) 205 over 256 bits (c) 410 over 512 bits

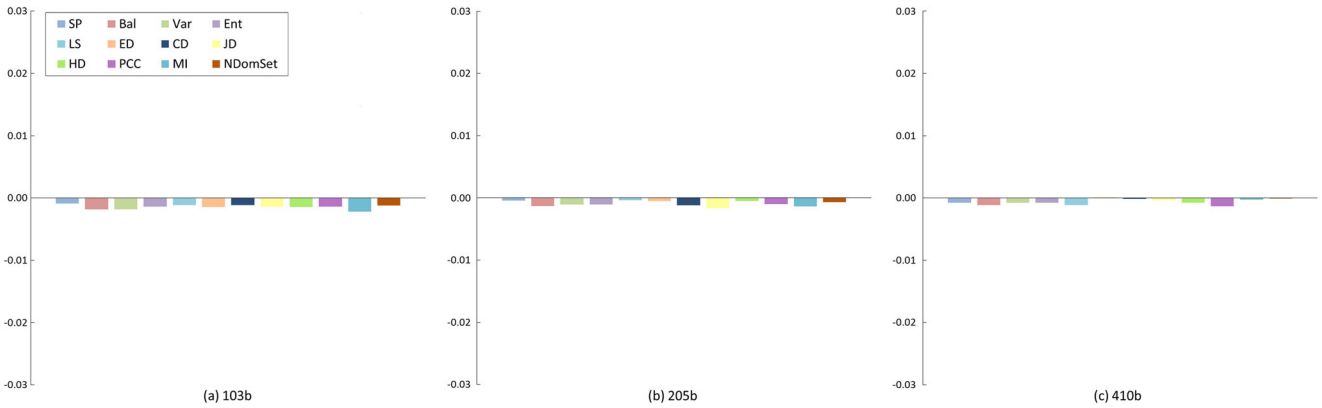

Fig. 3 The recall comparison of LSH to the bit selection methods with 80% hash bits on CIFAR-10, where the bars above (below) zero indicate that they are better (worse) than LSH. (a) 103 over 128 bits (b) 205 over 256 bits (c) 410 over 512 bits

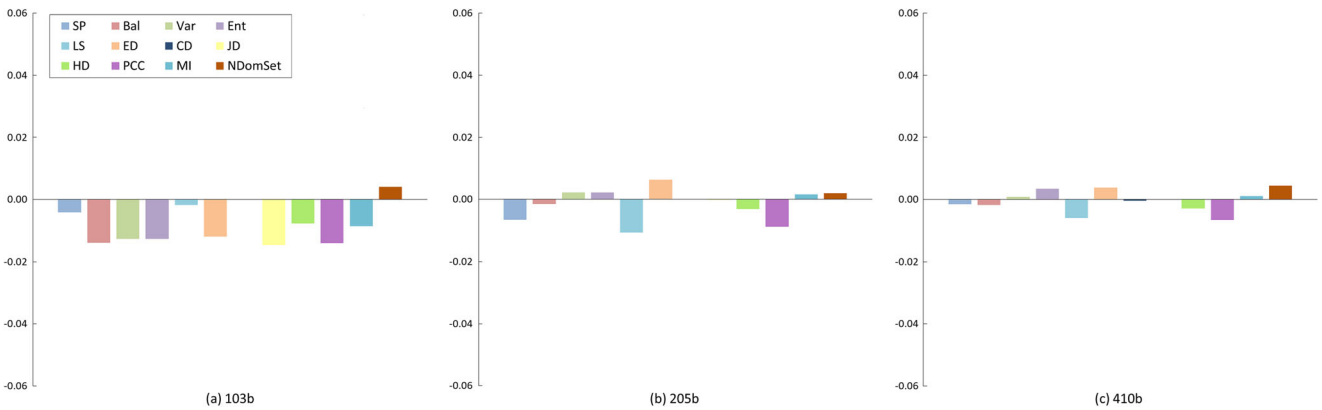

Fig. 4 The recall comparison of LSH to the bit selection methods with 80% hash bits on USPS, where the bars above (below) zero indicate that they are better (worse) than LSH. (a) 103 over 128 bits (b) 205 over 256 bits (c) 410 over 512 bits

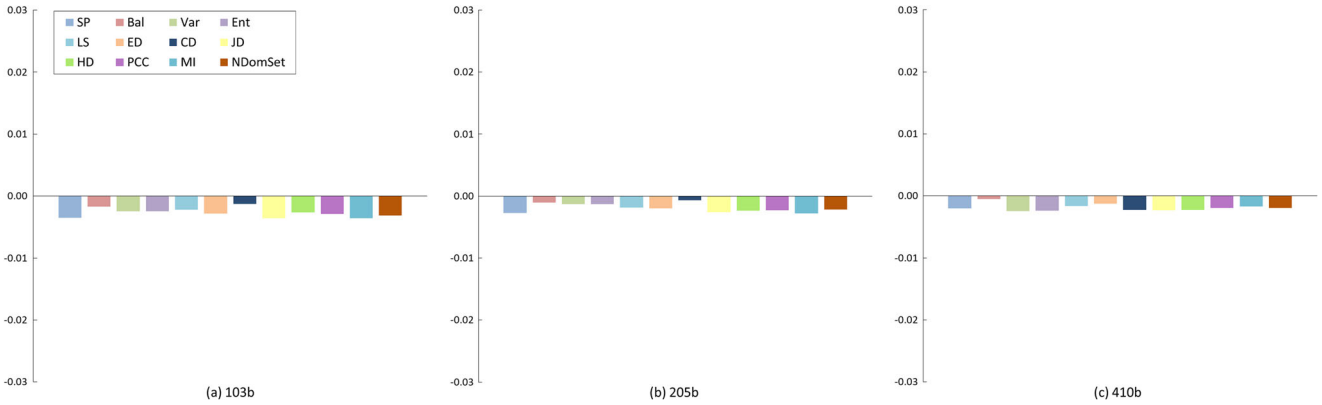

Fig. 5 The recall comparison of LSH to the bit selection methods with 80% hash bits on IAPR-TC12, where the bars above (below) zero indicate that they are better (worse) than LSH. (a) 103 over 128 bits (b) 205 over 256 bits (c) 410 over 512 bits

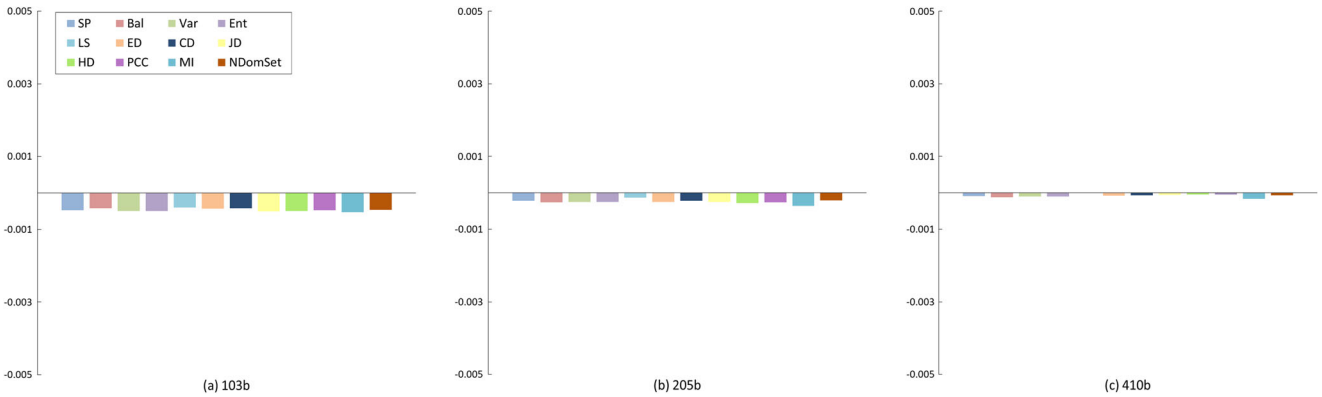

Fig. 6 The recall comparison of LSH to the bit selection methods with 80% hash bits on NUS-WIDE, where the bars above (below) zero indicate that they are better (worse) than LSH. (a) 103 over 128 bits (b) 205 over 256 bits (c) 410 over 512 bits

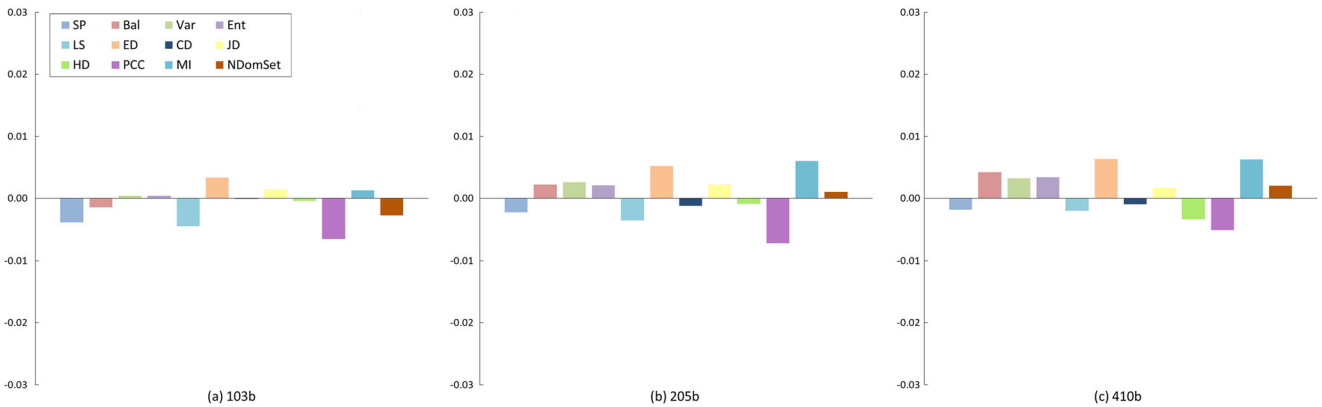

Fig. 7 The precision comparison of LSH to the bit selection methods with 80% hash bits on CALTECH101, where the bars above (below) zero indicate that they are better (worse) than LSH. (a) 103 over 128 bits (b) 205 over 256 bits (c) 410 over 512 bits

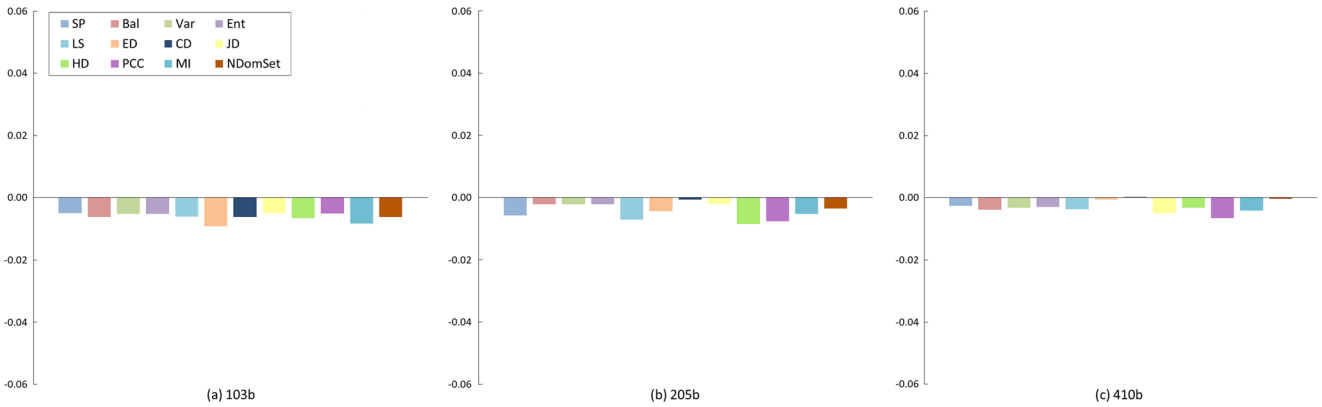

Fig. 8 The precision comparison of LSH to the bit selection methods with 80% hash bits on CIFAR-10, where the bars above (below) zero indicate that they are better (worse) than LSH. (a) 103 over 128 bits (b) 205 over 256 bits (c) 410 over 512 bits

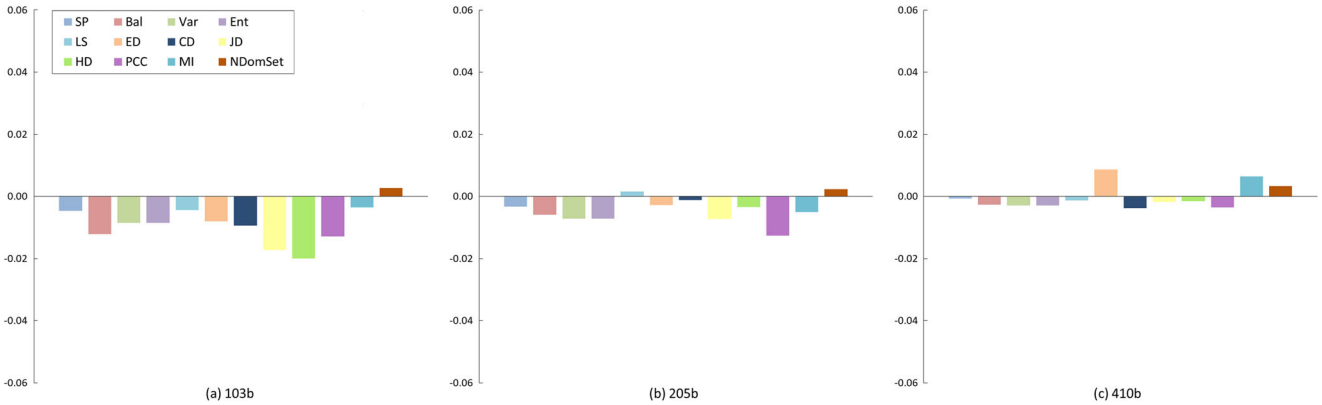

Fig. 9 The precision comparison of LSH to the bit selection methods with 80% hash bits on USPS, where the bars above (below) zero indicate that they are better (worse) than LSH. (a) 103 over 128 bits (b) 205 over 256 bits (c) 410 over 512 bits

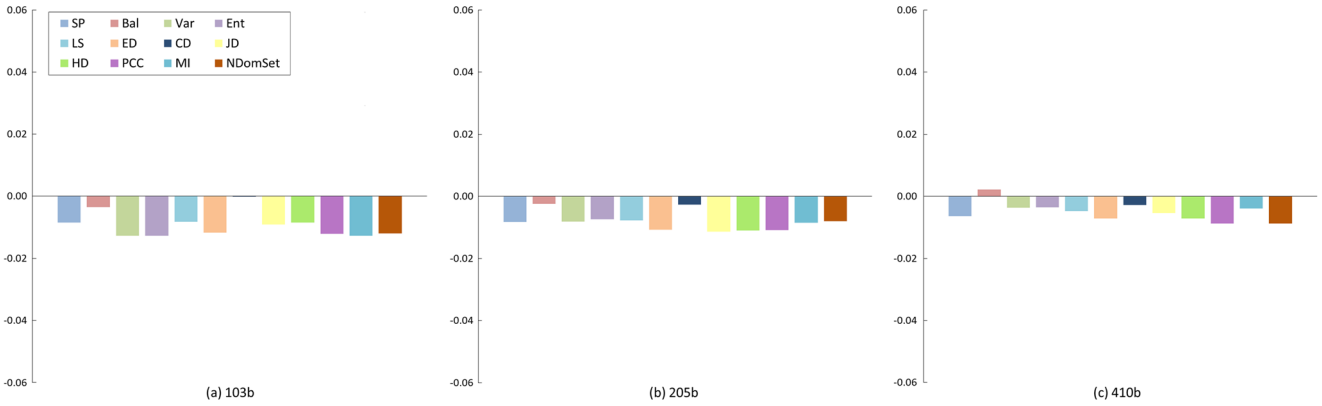

Fig. 10 The precision comparison of LSH to the bit selection methods with 80% hash bits on IAPR-TC12, where the bars above (below) zero indicate that they are better (worse) than LSH. (a) 103 over 128 bits (b) 205 over 256 bits (c) 410 over 512 bits

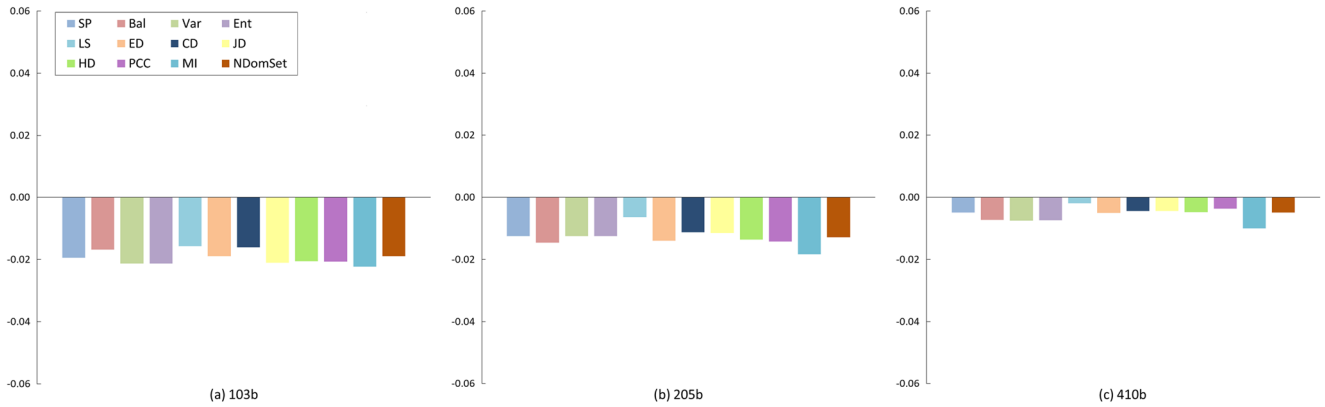

Fig. 11 The precision comparison of LSH to the bit selection methods with 80% hash bits on NUS-WIDE, where the bars above (below) zero indicate that they are better (worse) than LSH. (a) 103 over 128 bits (b) 205 over 256 bits (c) 410 over 512 bits

the selection methods with more bits have comparable performance to LSH.

5.3.2 Sensitivity of the bit selection methods

The second group of comparison experiments is conducted to testify the sensitivity of the bit selection methods with different numbers of hash bits selected. Given 512 hash bits generated via LSH on the data sets, we apply the bit selection methods to assembling new hash bits in various selection ratios, ranging form 50% to 100%. Afterwards, we estimated the mAP scores of these bit selection methods.

Figure 12 gives the mAP scores of the bit selection methods, where the x-axis denotes the bit selection ratio, ranging from 50% to 100% out of 512 hash bits generated by LSH. For clarity purpose, the mAP score of LSH is also provided and serves as the horizontal line. According to Fig. 12, it is observed that the bit selection methods with *Euclidean distance* (ED) and *mutual information* (MI) have better performance than LSH on CALTECH101, though

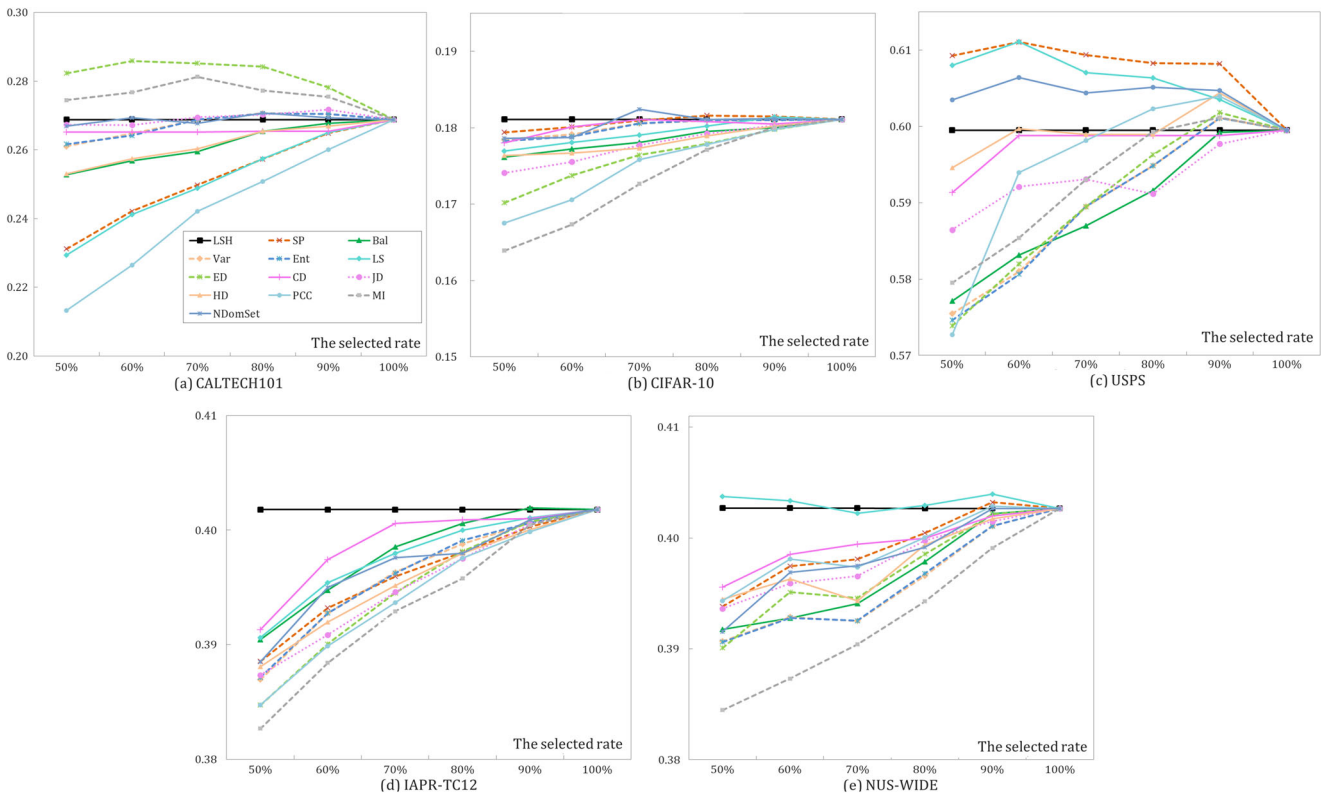

Fig. 12 The mAP scores of the bit selection methods with different quantities of bits, ranging from 50% to 100% of 512 bits

Table 1 The maximum reduction ratios (%) of the bit selection methods with less than 1% loss of mAP

	CALTECH101			CIFAR-10			USPS			IAPR-TC12			NUS-WIDE		
	128b	256b	512b	128b	256b	512b	128b	256b	512b	128b	256b	512b	128b	256b	512b
SP	18.75	20.00	20.89	**54.68**	**57.03**	**61.91**	37.50	45.31	60.90	29.63	29.67	43.35	39.84	47.26	50.50
Bal	29.68	44.51	60.93	32.02	35.54	44.43	15.63	23.04	37.30	37.50	41.60	50.00	33.59	36.71	40.42
Var	28.12	44.01	59.96	32.03	39.45	46.67	15.62	22.67	38.20	33.51	35.54	41.40	31.59	34.71	39.42
Ent	28.12	44.01	59.96	29.68	34.37	44.33	15.62	22.67	38.20	33.51	35.54	41.40	31.59	34.71	39.42
LS	17.92	17.18	20.00	54.68	57.03	56.44	28.12	41.20	53.12	48.20	49.21	**53.12**	**60.90**	**62.89**	**73.60**
ED	**39.84**	**55.07**	**67.77**	16.40	16.79	23.82	18.75	23.00	38.47	29.68	31.64	43.35	41.40	45.30	48.20
CD	39.06	43.35	58.98	46.87	53.13	55.07	33.59	42.57	46.28	**49.78**	**51.23**	52.53	56.25	52.30	55.85
JD	33.20	43.35	59.96	50.78	51.17	56.05	17.96	25.78	33.59	31.25	31.64	40.42	41.40	42.18	48.40
HD	18.75	27.73	24.80	49.21	55.07	55.07	20.00	26.95	33.10	30.86	31.74	42.38	46.09	46.09	50.00
PCC	14.02	13.10	14.06	41.40	48.04	57.03	21.87	25.78	33.20	30.95	31.74	41.99	46.87	43.35	51.95
MI	38.10	45.31	62.89	27.34	39.45	50.01	15.67	21.00	30.66	22.10	23.43	33.59	33.59	32.03	33.59
NDomSet	38.10	45.31	56.05	43.75	47.26	58.00	**55.46**	**72.65**	**86.32**	29.60	35.00	39.00	41.40	45.31	46.28

they contain less hash bits. Alternatively, the selection methods with *similarity preservation* SP and *Laplacian score* (LS) achieve higher scores than others, including NDomSet and LSH, on the USPS data set. Besides, the bit selection method involving *cosine distance* (CD) achieves stable performance even with less hash bits, around 60%, on the CALTECH101, CIFAR-10 and USPS data sets. Although the bit selection methods had relatively poor performance on the IAPR-TC12 and NUS-WIDE data sets, they are still competitive in the performance when the number of selected hash bits is around 90%.

5.3.3 Reduction ratio of the bit selection methods

The objective of the third group of experiments is to test the maximum reduction ratios of the bit selection methods, without degrading the performance significantly; that is, how many hash bits are needed to parallel in performance to LSH. Firstly, LSH is performed on the data sets to generate 128, 256 and 512 hash bits. Then the bit selection methods were applied to pick the least hash bits out so that the mAP scores of the selection methods were lower than those of LSH within 1%.

Table 1 records the maximum reduction ratios of the bit selection methods with less than 1% loss of mAP, where the bold values are the highest reduction ratios among the bit selection methods. From the experimental results in Table 1, it is obvious that the bit selection methods are capable of selecting less bits by removing redundant ones, without degrading the performance of LSH significantly. For example, the reduction ratios of the selection methods involving *Euclidean distance* (ED) are the best ones and reach to around 40%, 55% and 68% on the CALTECH101 data set, when the quantities of hash bits are 128, 256 and 512, respectively. Moreover, the maximum reduction ratios get greater as the number of candidate hash bits increases. This implies that a longer hash code often embodies more redundant information than a shorter one.

Table 2 lists the running time of selecting 205 over 256 hash bits, i.e., the reduction ratio is 20%, by the bit selection methods. As shown in Table 2, estimating the values of *balance* (Bal), *variance* (Var), *entropy* (Ent) and *Pearson correlation coefficient* (PCC) is more efficient than the other evaluation criteria. Besides, NDomSet is the most time consuming one among the selection methods, for it requires to calculate both mutual information and similarity preservation of hash bits. Even so, it is still acceptable for the large-scale data. This means that the proposed framework of bit selection has high efficiency and can be used as a post-processing stage for hashing techniques.

Table 2 The running time (seconds) of selecting 205 over 256 hash bits by the bit selection methods

	CALTECH101	CIFAR10	USPS	IAPR-TC12	NUS-WIDE
SP	0.859	0.710	0.705	1.030	1.05
Bal	0.016	0.101	0.016	0.033	0.372
Var	0.062	0.047	0.047	0.046	0.066
Ent	0.005	0.002	0.003	0.003	0.006
LS	0.259	0.217	0.220	0.238	0.256
ED	0.119	0.088	0.097	0.103	0.129
CD	0.115	0.120	0.098	0.167	0.154
JD	0.322	0.246	0.204	0.245	0.339
HD	0.343	0.421	0.152	0.320	0.362
PCC	0.130	0.123	0.115	0.075	0.092
MI	2.210	2.134	2.200	2.165	2.27
NDomSet	3.027	3.047	2.985	3.25	3.060

6 Conclusion

In this paper, a novel bit selection framework is introduced to choose important and representative bits out of the hash bits generated by hashing techniques. It mainly consists of three stages, i.e., bit generation, bit selection, and code assembly, wherein bit selection plays a central role. The stage of bit selection also involves three components, including selection strategies, bit evaluation, and bit ranking. To show the effectiveness of the proposed framework, we further exploit eleven evaluation criteria to measure the importance and similarity metrics of each bit generated by LSH, so that the bits with high importance and less similarity are selected to assemble new binary codes. We evaluated the proposed framework with the evaluation criteria on five commonly used data sets. Experimental results show the proposed bit selection framework works effectively in different cases, and the performance of LSH does not degrade significantly after redundant hash bits removed with the evaluation criteria.

In the future work, we will take other existing hashing techniques into account to testify that the proposed framework can work for general hashing techniques. Besides, advanced techniques of measuring correlation will also be considered to improve the reduction ratio of bit selection.

Acknowledgments The authors would like to thank the anonymous referees and the editors for their valuable comments and suggestions, helping to improve the paper significantly. This work was partially supported by the national NSF of China (NSFC) (61976195) and the NSF of Zhejiang Province (LY18F020019).

References

1. Wang J, Zhang T, Song J, Sebe N, Shen HT (2018) A survey on learning to hash. IEEE Trans Pattern Anal Mach Intell 40(4):769–790
2. Zhang B, Qian J (2021) Autoencoder-based unsupervised clustering and hashing. Appl Intell 51(1):493–505
3. Liu H, Li E, Liu X, Su K, Zhang S (2021) Anomaly detection with kernel preserving embedding. ACM Transactions on Knowledge Discovery from Data 15(5):91:1–91:
4. Zhang S, Li X, Zong M, Zhu X, Wang R (2018) Efficient knn classification with different numbers of nearest neighbors. IEEE Transactions on Neural Networks and Learning Systems 29(5):1774–1785
5. Quynh NH, Thuy QDT, Van CP, Van CN, Tao NQ (2018) An efficient image retrieval method using adaptive weights. Appl Intell 48(10):3807–3826
6. Silpa-Anan C, Hartley RI (2008) Optimised kd-trees for fast image descriptor matching. In: Proc. of the 2008 IEEE conf computer vision and pattern recognition (CVPR08), pp 24–26
7. Bentley JL (1975) Multidimensional binary search trees used for associative searching. Commun ACM 18(9):509–517
8. Nielsen F, Piro P, Barlaud M (2009) Tailored bregman ball trees for effective nearest neighbors. In: Proc of IEEE european workshop on computational geometry, pp 29–32
9. Liu H, Xu X, Li E, Li X, Zhang S (2021) Anomaly detection with representative neighbors. IEEE Transactions on Neural Networks and Learning Systems, 32(12). https://doi.org/10.1109/TNNLS.2021.3109898
10. Liu H, Li X, Zhang S, Tian Q (2020) Adaptive hashing with sparse matrix factorization. IEEE Transactions on Neural Networks and Learning Systems 31(10):4318–4329
11. Cai D (2021) A revisit of hashing algorithms for approximate nearest neighbor search. IEEE Trans Knowl Data Eng 33(6):2337–2348
12. Kulis B, Grauman K (2009) Kernelized locality-sensitive hashing for scalable image search. In: Proc of IEEE int conf computer vision (ICCV09), pp 2130–2137
13. Raginsky M, Lazebnik S (2009) Locality-sensitive binary codes from shift-invariant kernels. In: Proc of the annual conf neural information processing systems (NIPS09), pp 1509–1517
14. Chi L, Zhu X (2017) Hashing techniques: A survey and taxonomy. ACM Computting Surveys 50(1):1–36
15. Jégou H, Perronnin F, Douze M, Sánchez J, Pérez P, Schmid C (2012) Aggregating local image descriptors into compact codes. IEEE Trans Pattern Anal Mach Intell 34(9):1704–1716
16. Gong Y, Lazebnik S, Gordo A, Perronnin F (2013) Iterative quantization: a procrustean approach to learning binary codes for large-scale image retrieval. IEEE Trans Pattern Anal Mach Intell 35(12):2916–2929
17. Hoang T, Do TT, Nguyen TV, Cheung NM (2020) Unsupervised deep cross-modality spectral hashing. IEEE Trans Image Process 29:8391–8406
18. Liu W, Wang J, Kumar S, Chang SF (2011) Hashing with graphs. In: Proc of Int Conf Machine Learning (ICML2011), pp 1–8
19. Zhu X, Zhang L, Huang Z (2014) A sparse embedding and least variance encoding approach to hashing. IEEE Trans Image Process 23(9):3737–3750
20. Irie G, Li Z, Wu XM, Chang SF (2014) Locally linear hashing for extracting non-linear manifolds. In: Proc of the 2014 IEEE conf computer vision and pattern recognition (CVPR2014), pp 2123–2130
21. Pang QQ, Zhang L (2020) Fast backward iterative laplacian score for unsupervised feature selection. In: Proc of int conf knowledge science, engineering and management (KSEM2020), pp 409–420
22. Xie X, Chen H, Qian J (2019) Twin maximum entropy discriminations for classification. Appl Intell 49(6):2391–2399
23. Pan H, You X, Liu S, Zhang D (2021) Pearson correlation coefficient-based pheromone refactoring mechanism for multi-colony ant colony optimization. Appl Intell 51(2):752–774
24. Liu H, Sun J, Liu L, Zhang H (2009) Feature selection with dynamic mutual information. Pattern Recogn 42(7):1330–1339
25. Zhang D, Wang J, Cai D, Lu J (2010) Self-taught hashing for fast similarity search. In: Proc of the 33rd Int ACM SIGIR conf research and development in information, pp 18–25
26. Zhu X, Huang Z, Cheng H, Cui J, Shen HT (2013) Sparse hashing for fast multimedia search. ACM Trans Inf Syst 31(2):1–24
27. Liu X, He J, Chang SF (2017a) Hash bit selection for nearest neighbor search. IEEE Trans Image Process 26(11):5367–5380
28. Liu H, Liu L, Le TD, Lee I, Sun S, Li J (2017b) Nonparametric sparse matrix decomposition for cross-view dimensionality reduction. IEEE Transactions on Multimedia 19(8):1848–1859
29. Nie X, Jing W, Cui C, Zhang CJ, Zhu L, Yin Y (2020) Joint multi-view hashing for large-scale near-duplicate video retrieval. IEEE Trans Knowl Data Eng 32(10):1951–1965
30. Shen HT, Liu L, Yang Y, Xu X, Huang Z, Shen F, Hong R (2021) Exploiting subspace relation in semantic labels for cross-modal hashing. IEEE Trans Knowl Data Eng 33(10):3351–3365

31. Kong W, Li WJ, Guo M (2012) Manhattan hashing for large-scale image retrieval. In: Proc of the 35rd Int ACM SIGIR conf research and development in information, pp 45–54
32. Liu X, Nie X, Zhou Q, Nie L, Yin Y (2020) Model optimization boosting framework for linear model hash learning. IEEE Trans Image Process 29:4254–4268
33. Hasan MM, Srizon AY, Sayeed A, Hasan MAM (2021) High performance classification of caltech-101 with a transfer learned deep convolutional neural network. In: Proc of the 2021 IEEE int conf information and communication technology for sustainable development (ICICT4SD), pp 35–39
34. Gui J, Liu T, Sun Z, Tao D, Tan T (2018) Fast supervised fiscrete hashing. IEEE Trans Pattern Anal Mach Intell 40(2):490–496
35. Yan C, Bai X, Wang S, Zhou J, Hancock ER (2019) Cross-modal hashing with semantic deep embedding. Neurocomputing 337:58–66

Wenhua Zhou received the M.S degree in Zhejiang Normal University, Jinhua, China in 2022. He is currently a lectuer with the Department of Information Engineering, Jinhua Polytechnic, Jinhua, P.R. China. His research interests including image processing, information retrieval and data mining.

Huawen Liu received the M.S. degree and Ph.D. degree in computer science from Jilin University, Changchun, P.R. China, in 2007 and 2010, respectively. He is currently a full professor with the Department of Computer Science, Shaoxing University, Shaoxing, P.R. China. He was a Visiting Scholar with the department of Computer Science at The University of Texas at San Antonio from 2018 to 2019, and at University of South Australia from 2012 to 2013. His current research interests include artificial intelligence, machine learning and data mining.

Jungang Lou received the B.S. degree in Mathematics from Zhejiang Normal University, China, in 2003, and the M.S. degree in computational mathematics and the Ph.D. degree in computer science and technology from Tongji University, Shanghai, China, in 2006 and 2010, respectively. He is currently a Professor with the School of Information Engineering, Huzhou University, Huzhou, China. He was a Visiting Scholar with the department of Computer Science at The University of Texas at San Antonio between Nov. 2017 and May 2018 (advisor Professor Qi Tian, IEEE Fellow). His current research interests include artificial intelligence, dependable computing and information security. He has published over 100 papers in refereed international journals including IEEE TC, IEEE TCAD, IEEE TNNLS, IEEE TR, IEEE TCYB, IEEE TCAS-II, Pattern Recognition, Information Sciences and so on.

Xin Chen received the M.S. and Ph.D. degrees in computer science from National University of Defense Technology, Changsha, China in 2005 and 2011, respectively. She is an Associate Professor with the Department of Computer Science, Zhejiang Normal University, Jinhua, China. Her research interests include medical image processing, computer vision, and machine learning. She has published over 50 papers in refereed international journals and international conferences.