Binary Code Ranking with Weighted Hamming Distance

Lei Zhang[1,2], Yongdong Zhang[1], Jinhui Tang[3], Ke Lu[2], Qi Tian[4]
[1]Institute of Computing Technology, Chinese Academy Sciences, No.6 Kexueyuan South Road, Beijing, China
[2]University of Chinese Academy of Sciences, No.19A Yuquan Road, Beijing, China
[3]Nanjing University of Science and Technology, Xiaolingwei 200, Nanjing, China
[4]University of Texas at San Antonio, Texas, TX, USA
{zhanglei09, zhyd}@ict.ac.cn, tangjh1981@acm.org, luk@gucas.ac.cn, qitian@cs.utsa.edu

Abstract

Binary hashing has been widely used for efficient similarity search due to its query and storage efficiency. In most existing binary hashing methods, the high-dimensional data are embedded into Hamming space and the distance or similarity of two points are approximated by the Hamming distance between their binary codes. The Hamming distance calculation is efficient, however, in practice, there are often lots of results sharing the same Hamming distance to a query, which makes this distance measure ambiguous and poses a critical issue for similarity search where ranking is important. In this paper, we propose a weighted Hamming distance ranking algorithm (WhRank) to rank the binary codes of hashing methods. By assigning different bit-level weights to different hash bits, the returned binary codes are ranked at a finer-grained binary code level. We give an algorithm to learn the data-adaptive and query-sensitive weight for each hash bit. Evaluations on two large-scale image data sets demonstrate the efficacy of our weighted Hamming distance for binary code ranking.

1. Introduction

High-dimensional similarity search is a fundamental problem in many content-based search systems [20, 23] and it has been widely used in many related application areas, such as machine learning, computer vision and data mining. To solve this problem efficiently, many methods have been proposed, such as KD-Tree [2][18] and Locality Sensitive Hashing (LSH) [1]. Recently, binary hashing [20, 21, 16, 14, 12, 3, 6, 11] is becoming increasingly popular for efficient approximated nearest neighbor (ANN) search due to its good query and storage efficiency.

The goal of binary hashing is to learn binary representations for data such that the neighborhood structure in the original data space can be preserved after embedded into Hamming space. Given a dataset, binary hashing generates binary code for each data point and approximates the distance or similarity of two points by the Hamming distance between their binary codes, which means most hashing methods rank the returned results based on their Hamming distances to query. This distance measure is widely used because of the calculation efficiency. However, since the Hamming distance is discrete and bounded by the code length, in practice, there will be a lot of data points sharing the same Hamming distance to the query and the ranking of these data points is ambiguous, which poses a critical issue for similarity search, *e.g.* k nearest neighbor search, where ranking is important. As a result, most existing binary hashing methods lack in providing a good ranking of results.

In this paper, we propose a weighted Hamming distance ranking algorithm (WhRank) to improve the ranking performance of binary hashing methods. By assigning different bit-level weights to different hash bits, it is possible to rank two binary codes sharing the same Hamming distance to a query at a finer-grained binary code level, and gives binary hashing methods the ability to distinguish between the relative importance of different bits. We also give an algorithm to learn a set of dynamic bit-level weights of hash bits for a given query. By taking account of the information provided by the hash functions and dataset, we learn a set of data-adaptive and query-sensitive bit-level weights to reveal the relative importance of different hash bits.

The rest of this paper is organized as follows. The related work is discussed in Section 2. The weighted Hamming distance ranking algorithm is proposed in Section 3 and analyzed in Section 4. Section 5 describes our experiments and Section 6 concludes this paper.

2. Related Work

With the proliferation of various kinds of data, e.g. music, image and video, in content-based search systems, fast similarity search has attracted a significant attention. One

1063-6919/13 $26.00 © 2013 IEEE
DOI 10.1109/CVPR.2013.208

classical kind of methods to address this problem is the tree-based index, such as KD-Tree [2][18]. However, this kind of methods cannot work well for high-dimensional data since the performance degrade significantly to linear scan as the dimensionality increases. Recently, hashing based methods [5, 3, 24] has been widely used for efficient similarity search in a large variety of applications due to its efficiency in terms of query speed and storage space. The goal of binary hashing is to map each dataset point to a compact binary code such that similar data points in the original data space can be mapped to similar binary codes in Hamming space. One of the representative methods is Locality Sensitive Hashing (LSH) [1] and its variants [15, 9, 14, 6]. Theoretically, LSH-related methods usually require long code to achieve good precision. However, long code results in low recall since the collision probability of similar points mapped to similar binary codes decreases exponentially as the code length increases. As a result, LSH-related methods usually construct multi-tables to ensure a reasonable probability that a query will collide with its near neighbors in at least one of the tables, which leads to a long query time and increases the memory occupation. To generate more compact binary code, many algorithms have been proposed. Semantic Hashing [17] adopts a deep generative model based on restricted Boltzmann machine to learn the hash functions that map similar points to similar binary codes. Spectral Hashing (SPH) [21] uses spectral graph partitioning strategy for hash function learning and uses the simple analytical eigenfunction solution of 1-D Laplacians as the hash function. In PCA-Hashing (PCAH)[20], the eigenvectors corresponding to the largest eigenvalues of the dataset covariance matrix are used to get binary codes. In [3], Iterative Quantization (ITQ) is proposed to learn an orthogonal rotation matrix to refine the initial PCA-projection matrix of PCAH to minimize the quantization error of mapping the data from original data space to Hamming space. To minimize the reconstruction error between the distances in the original data space and the Hamming distances of the corresponding binary codes, the Binary Reconstruction Embedding (BRE) is proposed in [8]. Moreover, to exploit the spectral properties of the data affinity to generate better binary codes, many other algorithms, such as Semi-Supervised Sequential Projection Hashing [19], Anchor Graph Hashing [12] and Kernel-Based Supervised Hashing [11] have been developed and give commendable search performances.

In most existing binary hashing methods, including those methods discussed above, the returned results of a given query are simply ranked based on their Hamming distance to the query. The calculation of Hamming distance is efficient, however, since this distance metric gives each hash bit the same weight, it unable to distinguish between the relative importance of different bits and causes ambiguity for ranking. One way to alleviate this ambiguity is assigning different bit-level weights to different hash bits. The weighted Hamming distance has been used for image retrieve, including Hamming distance weighting [4] and the AnnoSearch [20]. In [20], each bit of the binary code is assigned with a bit-level weight, while in [4], the aim is to weight the overall Hamming distance of local features for image matching. In these works, only a single set of weights is used to measure either the importance of each bit in Hamming space [20], or to rescale the Hamming distance for better image matching [4]. In [7], Jiang *et al*. propose a query-adaptive Hamming distance for image retrieve which assigns dynamic weights to hash bits, such that each bit is treated differently and dynamically. They harness a set of semantic concept classes that cover most semantic elements of image content. Then, different weights for each of the classes are learned with a supervised learning algorithm. To compute the bit-level weights for a given query, a k nearest neighbor search is performed based on the original Hamming distance first, then a linear combination of the weights of classes contained in the result list is used as the query-adaptive weights.

In [22], the authors propose a query-sensitive hash code ranking algorithm (QsRank) for PCA-based hashing methods. Given a query, QsRank assigns two weights to each hash bit and defines a score function to measure the confidence of the neighbors of a query mapped to a binary code. The returned codes are ranked based on the their scores. Experimental results demonstrate the efficacy of QsRank. There are three key differences between QsRank and our method WhRank. First, QsRank is developed only for PCA-based binary hashing while WhRank can be applied to most existing binary hashing methods. Second, QsRank is developed for ϵ-neighbor search, which requires the structure of the original data space maintained well after dimension reduction, while WhRank does not. Third, QsRank makes a strong assumption about the data distribution (uniform distribution), while WhRank only makes an assumption about the distribution of the differences between a query and its neighbors, which is more appropriate in most cases.

3. Ranking with Weighted Hamming Distance

In this section, we present the weighted Hamming distance ranking algorithm. In most binary hashing algorithms, the distance between two points is simply measured by the Hamming distance between their binary codes. This distance metric is somewhat ambiguous, since for a K-bits binary code $H(\boldsymbol{p})$, there are $\binom{K}{m}$ different binary codes sharing the same distance m to $H(\boldsymbol{p})$. In most binary hashing algorithms, each hash bit takes the same weight and makes the same contribution to distance calculation. On the contrary, in our algorithm, we give different bits different weights. With the bit-level weights, the returned binary codes can be ranked by the weighted Hamming distance at

a finer-grained binary code level rather than at the original integer Hamming distance level. The bit-level weight associated with hash bit k is denoted as ω_k. In the following, we will show that an effective bit-level weight is not only data-dependent, but also query-dependent. Note that, our algorithm is not to propose a new binary hashing method, but to give a ranking algorithm to improve the search accuracy of most existing binary hashing methods. Some notations are given below to facilitate our discussion.

Given a dataset $\mathcal{X} = \{\boldsymbol{x}^{(i)}\}_{i=1}^{N} \in \mathbb{R}^d$, the neighbor set of $\boldsymbol{x}$ is denoted as $N(\boldsymbol{x})$. The paradigm of binary hashing is to first use a set of linear or non-linear hash functions $F = \{f_k : \mathbb{R}^d \to \mathbb{R}\}_{k=1}^{K}$ to map $\boldsymbol{x} \in \mathcal{X}$ to $F(\boldsymbol{x}) \in \mathbb{R}^K$, and then binarize $F(\boldsymbol{x}) = (f_1(\boldsymbol{x}), \cdots, f_K(\boldsymbol{x}))^T$ by comparing each $f_k(\boldsymbol{x})$ with a threshold T_k to get a K-bit binary code $H(\boldsymbol{x}) \in \{0,1\}^K$. Hence, the binary hash function is $h_k(\boldsymbol{x}) = \text{sgn}(f_k(\boldsymbol{x}) - T_k)$. We call $f_k(\boldsymbol{x})$ the *unbinarized* hash value. Each dimension of $H(\boldsymbol{x})$ is called a hash bit, and for a query $\boldsymbol{q}$ and its neighbor $\boldsymbol{p}$, if the k-th bit of $H(\boldsymbol{q})$ and $H(\boldsymbol{p})$ is different, we call there is a bit-flipping on hash bit k. The weighted Hamming distance between two binary codes $\boldsymbol{h}^{(1)}$ and $\boldsymbol{h}^{(2)}$ is denoted as $D_H^w(\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)})$.

3.1. Data-Adaptive Weight

We introduce a term *discriminating power* to denote the ability of a hash function $h_k(\boldsymbol{x})$ mapping similar data points to the same bit (0/1). A hash function $h_k(\boldsymbol{x})$ is called discriminative if the probability of similar data points mapped to the same bit by $h_k(\boldsymbol{x})$ is not small (> 0.5). The more discriminative $h_k(\boldsymbol{x})$, the more discriminative hash bit k. Obviously, the discriminating power of a hash function is dependent on the algorithm generates it and the dataset used for training. In many binary hashing methods, *e.g.* PCAH [20], SPH [21], ITQ [3] and AGH [12], the discriminating power of different hash function is intrinsically different. For a hash function with a stronger discriminating power, it's less likely for this hash function to generate different bits for two neighbor points. In other word, for a query $\boldsymbol{q}$ and two data points $\boldsymbol{p}^{(1)}$, $\boldsymbol{p}^{(2)}$ sharing the same Hamming distance (1) to $\boldsymbol{q}$, where $H(\boldsymbol{p}^{(1)})$ and $H(\boldsymbol{p}^{(2)})$ are different with $H(\boldsymbol{q})$ on hash bits k_1 and k_2 respectively. If hash bit k_1 is more discriminative than k_2, then $\boldsymbol{p}^{(1)}$ is considered to be less similar with $\boldsymbol{q}$ than $\boldsymbol{p}^{(2)}$, since the bit-flipping on hash bit k_1 gives a higher confidence that $\boldsymbol{p}^{(1)}$ is not a neighbor of $\boldsymbol{q}$ than that on k_2. To make $D_H^w(H(\boldsymbol{q}), H(\boldsymbol{p}^{(1)}))$ larger than $D_H^w(H(\boldsymbol{q}), H(\boldsymbol{p}^{(2)}))$, ω_{k_1} should be larger than ω_{k_2}, which means the more discriminative a hash bit k is, the larger the associated weight ω_k is.

As the discriminating power of a hash bit k (*i.e.* hash function $h_k(\boldsymbol{x})$) is related to the probability of similar points mapped to the same bit by $h_k(\boldsymbol{x})$, given a hash function $h_k(\boldsymbol{x})$, we can use the distribution of $h_k(\boldsymbol{p}) - h_k(\boldsymbol{q})$, where $\boldsymbol{p} \in N(\boldsymbol{q})$, to reveal how discriminative hash bit k is.

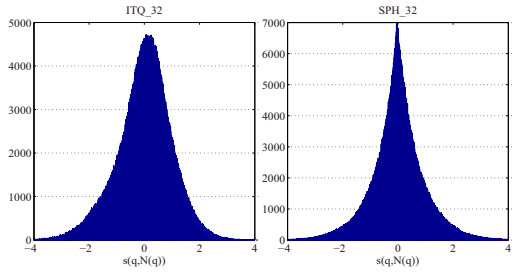

Figure 1. Histograms of the differences between the unbinarized hash values of a query and its neighbors, generated by ITQ [3] and SPH [21]. Dataset for illustration is ANN-SIFT1M [5].

However, since $h_k(\boldsymbol{p})$ and $h_k(\boldsymbol{q})$ are binarized, too much useful information is lost. An alternative is to use the distribution of the difference between their unbinarized hash values, *i.e.* $s_k(\boldsymbol{p}, \boldsymbol{q}) = f_k(\boldsymbol{p}) - f_k(\boldsymbol{q})$, to reveal the discriminating power. If $s_k(\boldsymbol{p}, \boldsymbol{q})$ is distributed in a small interval centered around 0, then the probability of $h_k(\boldsymbol{p}) = h_k(\boldsymbol{q})$ is high, yielding a high discriminative hash bit k. Fig. 1 gives the distribution of $s_k(\boldsymbol{p}, \boldsymbol{q})$ for ITQ [3] and SPH [21] using 32 bits binary code. As can be seen from this figure, the distributions are all Bell-shaped, as all these binary hashing methods try to minimize the distances between similar points after hashing. As a result, ω_k is a function of the distribution of s_k, which is parameterized by the its mean μ_k and standard deviation σ_k:

$$\omega_k = g(\mu_k, \sigma_k) \tag{1}$$

Note that, hash bit k is more discriminative if σ_k is smaller, therefore, $\omega_k = g(\mu_k, \sigma_k)$ should be monotonically non-increasing w.r.t. σ_k. An illustration for ITQ is given in the Right of Fig. 2, as shown, the probability of bit-flipping on hash bit k increases with the standard deviation σ_k.

3.2. Query-Sensitive Weight

Meanwhile, for a specified data point $\boldsymbol{q}$, the probability of its neighbor $\boldsymbol{p}$ mapped to a bit different from $H_k(\boldsymbol{q})$ by hash function $h_k(\boldsymbol{x})$ is also dependent on $\boldsymbol{q}$ itself. Intuitively, if $|f_k(\boldsymbol{q}) - T_k|$ is small, then after adding a random noise $\tilde{\boldsymbol{n}}$ to $\boldsymbol{q}$, it's more likely that $f_k(\boldsymbol{q} + \tilde{\boldsymbol{n}})$ lies on the opposite side of T_k as compared to $f_k(\boldsymbol{q})$, which means the probability of $h_k(\boldsymbol{p})$ differs from $h_k(\boldsymbol{q})$ for $\boldsymbol{p} \in N(\boldsymbol{q})$ is high. A simple example of this intuition is shown in the Left of Fig. 3. A query $\boldsymbol{q}$ is mapped to "101", the binary codes of $\boldsymbol{p}_1$, $\boldsymbol{p}_2$ and $\boldsymbol{p}_3$ are "001", "111", "110" respectively. Based on the Hamming distance, the result list of $\boldsymbol{q}$ is "001", "111", "110". However, it's more suitable to rank the hash code "110" and "111" before "001" because $\boldsymbol{q}$ is far from the threshold of hash function h_1, it's less likely for a near neighbor of $\boldsymbol{q}$ lies on the opposite side of h_1. Moreover, as shown in the Left of Fig. 2, the probability of

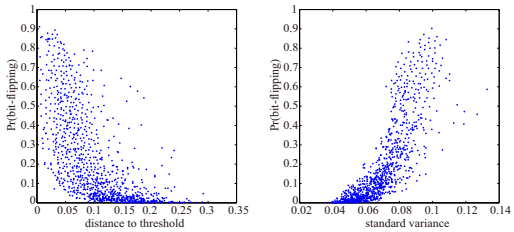

Figure 2. The probability of a query $\boldsymbol{q}$'s neighbor, $\boldsymbol{p}$, mapped to a bit different from $h(\boldsymbol{q})$ by hash function $h(\boldsymbol{x}) = \text{sgn}(f(\boldsymbol{x}) - T)$. The abscissa of the Left is $|f(\boldsymbol{q}) - T|$, and the abscissa of the Right is the standard variance of the distribution of $f(\boldsymbol{p}) - f(\boldsymbol{q})$.

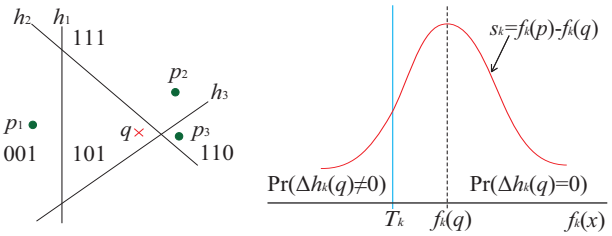

Figure 3. The Left gives an example where Hamming distance causes ambiguity for binary code ranking. The Right illustrates the probability of a neighbor of $\boldsymbol{q}$ mapped to a different bit by hash function $f_k(x)$, T_k is the binary threshold.

bit-flipping increases with $|f_k(\boldsymbol{q}) - T_k|$ for the most part. As a result, ω_k (1) is not only dependent on the hash function $h_k(\boldsymbol{x})$, but also dependent on the specified data point $\boldsymbol{q}$, and on this account, ω_k is also a function of $\boldsymbol{q}$. As a result, eq. (1) can be rewritten as:

$$\omega_k(\boldsymbol{q}) = g(\mu_k, \sigma_k, \boldsymbol{q}) \tag{2}$$

Moreover, the smaller $|f_k(\boldsymbol{q}) - T_k|$, the larger the probability of bit-flipping on hash bit k, thus the smaller ω_k. Therefore, $\omega_k(\boldsymbol{q})$ should also be monotonically non-decreasing w.r.t. $|f_k(\boldsymbol{q}) - T_k|$.

3.3. Dynamic Bit-level Weighting

In the previous sections 3.1 and 3.2, we show that an effective bit-level weight is not only data-dependent, but also query-dependent. In this section, we give a simple method to calculate the data-adaptive and query-sensitive bit-level weight $\omega_k(\boldsymbol{q})$ of each hash bit k for a given query $\boldsymbol{q}$, and we will show that $\omega_k(\boldsymbol{q})$ satisfies the abovementioned constraints theoretically. The intuition behind our method is: given a query $\boldsymbol{q}$ and two binary codes $\boldsymbol{h}^{(1)}, \boldsymbol{h}^{(2)}$, after adding a random noise $\tilde{\boldsymbol{n}}$ to $\boldsymbol{q}$, if the probability of $H(\boldsymbol{q} + \tilde{\boldsymbol{n}}) = \boldsymbol{h}^{(1)}$, denoted as $\Pr(\boldsymbol{h}^{(1)}|H(\boldsymbol{q}))$, is larger than $\Pr(\boldsymbol{h}^{(2)}|H(\boldsymbol{q}))$, then the data points mapped to $\boldsymbol{h}^{(1)}$ are considered to be more similar neighbors of $\boldsymbol{q}$ rather than those mapped to $\boldsymbol{h}^{(2)}$, which means the weighted Hamming distance $D_H^w(H(\boldsymbol{q}), \boldsymbol{h}^{(1)})$ is smaller than $D_H^w(H(\boldsymbol{q}), \boldsymbol{h}^{(2)})$. Therefore, given a query $\boldsymbol{q}$ and a binary code $\boldsymbol{h}$, a function parameterized by $\Pr(\boldsymbol{h}|H(\boldsymbol{q}))$ is used as a probabilistic interpretation of $D_H^w(H(\boldsymbol{q}), \boldsymbol{h})$. This function should be monotonically non-increasing w.r.t. $\Pr(\boldsymbol{h}|H(\boldsymbol{q}))$. Furthermore, if $\Pr(\boldsymbol{h}|H(\boldsymbol{q})) \approx 1$, $D_H^w(H(\boldsymbol{q}), \boldsymbol{h})$ should be small, and if $\Pr(\boldsymbol{h}|H(\boldsymbol{q})) \approx 0$, $D_H^w(H(\boldsymbol{q}), \boldsymbol{h})$ should be relatively large. A famous function satisfies these constraints is the Information Entropy. As a result, given a query $\boldsymbol{q}$, the weighted Hamming distance between $H(\boldsymbol{q})$ and a binary code $\boldsymbol{h}$ is defined as follows:

$$D_H^w(H(\boldsymbol{q}), \boldsymbol{h}) \approx -\log \Pr(\boldsymbol{h}|H(\boldsymbol{q})) \tag{3}$$

Assume all the hash bits are independent [22], we have:

$$\begin{aligned} \Pr(\boldsymbol{h}|H(\boldsymbol{q})) \quad = \quad & \prod_k \Pr(\boldsymbol{h}_k \neq h_k(\boldsymbol{q})) * \\ & \prod_k \Pr(\boldsymbol{h}_k = h_k(\boldsymbol{q})) \end{aligned} \tag{4}$$

where $\Pr(\boldsymbol{h}_k \neq h_k(\boldsymbol{q}))$ (denoted by $\Pr(\Delta h_k(\boldsymbol{q}) \neq 0)$) is the probability of hash bit k of $\boldsymbol{h}$ flipped as compared with that of $H(\boldsymbol{q})$, and $\Pr(\boldsymbol{h}_k = h_k(\boldsymbol{q}))$ (denoted by $\Pr(\Delta h_k(\boldsymbol{q}) = 0)$) is the probability of hash bit k of $\boldsymbol{h}$ not flipped . Apparently, these two probabilities are dependent on the specified query $\boldsymbol{q}$ and the hash function $h_k(\boldsymbol{x})$.

Since the weighted Hamming distance is used for ranking, the ranking of each $D_H^w(H(\boldsymbol{q}), \boldsymbol{h})$ is more crucial than their actual values. Therefore, by dividing each $\Pr(\boldsymbol{h}|H(\boldsymbol{q}))$ by $\prod_{k=1}^K \Pr(\Delta h_k(\boldsymbol{q}) = 0)$ without changing the ranking of each $D_H^w(H(\boldsymbol{q}), \boldsymbol{h})$, we get a modified weighted Hamming distance:

$$D_H^w(H(\boldsymbol{q}), \boldsymbol{h}) = \sum_{k \in S} \lambda_k(\boldsymbol{q}) \tag{5}$$

where S is the set of hash bits in $\boldsymbol{h}$ differ from $H(\boldsymbol{q})$, and

$$\lambda_k(\boldsymbol{q}) = \log \frac{\Pr(\Delta h_k(\boldsymbol{q}) = 0)}{\Pr(\Delta h_k(\boldsymbol{q}) \neq 0)} = \log \frac{1 - \Pr(\Delta h_k(\boldsymbol{q}) \neq 0)}{\Pr(\Delta h_k(\boldsymbol{q}) \neq 0)} \tag{6}$$

Equation (6) is a monotonically decreasing function w.r.t. $\Pr(\Delta h_k(\boldsymbol{q}) \neq 0)$. The smaller $\Pr(\Delta h_k(\boldsymbol{q}) \neq 0)$, the smaller the probability of a data point $\boldsymbol{p} \in N(\boldsymbol{q})$ mapped to a different bit by $h_k(\boldsymbol{x})$, thus the more discriminative hash bit k. Therefore, $\lambda_k(\boldsymbol{q})$ satisfies the constraints for data-adaptive weight introduced in Section 3.1.

To calculate $\Pr(\Delta h_k(\boldsymbol{q}) \neq 0)$ or $\Pr(\Delta h_k(\boldsymbol{q}) = 0)$, the distribution of $h_k(\boldsymbol{q} + \tilde{\boldsymbol{n}}) - h_k(\boldsymbol{q})$ is essential. Based on our discussion in Section 3.2, we can use the distribution of $s(\boldsymbol{q} + \tilde{\boldsymbol{n}}, \boldsymbol{q}) = f_k(\boldsymbol{q} + \tilde{\boldsymbol{n}}) - f_k(\boldsymbol{q})$ with density function $\text{pdf}_k(s)$ to estimate $\Pr(\Delta h_k(\boldsymbol{q}) \neq 0)$. The Right of Fig. 3 shows the probability of a neighbor of $\boldsymbol{q}$, $\boldsymbol{p}$, mapped to a bit

different from $h_k(\boldsymbol{q})$. If $f_k(\boldsymbol{q}) > T_k$, we have:

$$\begin{aligned}\Pr(\Delta h_k(\boldsymbol{q}) \neq 0) &= \Pr(f_k(\boldsymbol{p}) < T_k) \\ &= \Pr(s_k(\boldsymbol{p},\boldsymbol{q}) \leq T_k - f_k(\boldsymbol{q})) \\ &= \int_{-\infty}^{T_k - f_k(\boldsymbol{q})} \text{pdf}_k(s)ds \end{aligned} \quad (7)$$

The Gaussian distribution assumption for pdf_k is used for PCAH, LSH and ITQ in our experiments, since they are all Gaussian-like distributions as shown in Fig. 1. Therefore, $\text{pdf}_k(s) = \mathcal{N}(\mu_k, \sigma_k)$, and if $f_k(\boldsymbol{q}) > T_k$, we have:

$$\Pr(\Delta h_k(\boldsymbol{q}) \neq 0) = \frac{1}{2}\left[1 + \text{erf}(\frac{T_k - f_k(\boldsymbol{q}) - \mu_k}{\sigma_k\sqrt{2}})\right] \quad (8)$$

Similarly, for $f_k(\boldsymbol{q}) < T_k$:

$$\Pr(\Delta h_k(\boldsymbol{q}) \neq 0) = \frac{1}{2}\left[1 - \text{erf}(\frac{T_k - f_k(\boldsymbol{q}) - \mu_k}{\sigma_k\sqrt{2}})\right] \quad (9)$$

where erf is the Gauss error function. For SPH and AGH, we use the Laplace distribution assumption for pdf_k, thus $\text{pdf}_k = \exp\{|x - \mu_k|/b_k\}/2b_k$, where $b_k = \sigma_k/\sqrt{2}$.

In our experiments, we set $\omega_k(\boldsymbol{q}) = \lambda_k(\boldsymbol{q})$ and denote this weighting scheme as **WhRank**. Given a query $\boldsymbol{q}$, first the unbinarized hash value $f_k(\boldsymbol{q})$ of each hash bit k is calculated. Then, the adaptive weight $\omega_k(\boldsymbol{q})$ is calculated using eq. (8)(9)(6). Apparently, the larger σ_k, the smaller $\omega_k(\boldsymbol{q})$. Moreover, the smaller $|f_k(\boldsymbol{q}) - T_k|$, the larger $\Pr(\Delta h_k(\boldsymbol{q}) \neq 0)$, thus the smaller $\omega_k(\boldsymbol{q})$. Therefore, $\omega_k(\boldsymbol{q})$ satisfies the constraints for data-adaptive and query-sensitive weight introduced in Section 3.1 and 3.2. For the Laplace distribution assumption and the Student's t-distribution assumption used in our experiments, these discussions still hold. Another straightforward dynamic bit-level weighting is setting $\omega_k(\boldsymbol{q}) = |T_k - f(\boldsymbol{q})|/\sigma_k$. In our experiments, we use this weighting scheme as a natural baseline and denote it as **WhRank1**. Note that, since we make no assumption about the hashing method used in the bit-level weights learning, our algorithm, WhRank, can be applied to different kinds of hashing methods.

4. Analysis

As shown in eq. (5), given a query $\boldsymbol{q}$ and a binary code $\boldsymbol{h}$, $D_H^w(H(\boldsymbol{q}), \boldsymbol{h})$ can be calculated efficiently as: $\boldsymbol{\omega}^T(\boldsymbol{q})(H(\boldsymbol{q}) \otimes \boldsymbol{h})$, where $\otimes$ means the **xor** of two binary codes and $\boldsymbol{\omega}(\boldsymbol{q}) = (\omega_1(\boldsymbol{q}), \omega_2(\boldsymbol{q}), \cdots, \omega_K(\boldsymbol{q}))^T$. While the weighted distances can now be calculated by inner-product operation, it is actually possible to avoid this computational cost by computing the traditional Hamming distance first, and then ranking the returned binary codes based on their weighted-Hamming distances to $H(\boldsymbol{q})$. Therefore, the ranking of the returned binary codes can be obtained with minor additional cost.

To learn the μ_k and σ_k of a hash function $h_k(\boldsymbol{x})$, we construct a training set consists of s query points, each of which has m neighbors. The complexity of calculating the unbinarized hash values of each query and its neighbors is almost $O(s(m+1)d)$, and the complexity of calculating μ_k and σ_k is bounded by $O(3sm)$. Therefore, the overall training complexity of our parameters learning stage is bounded by $O(K * s(md + d + 3m)) \approx O(Ksmd)$.

5. Experiments

5.1. Experimental Setup

Our experiments are carried out on two benchmark datasets: MINST70K and ANN-SIFT1M. The MNIST70K [10] consists of 70K 784-dimensional images, each of which is associated with a digit label from '0' to '9', and is split into a database set (*i.e.* training set, 60K) and a query set (10K). The ANN-SIFT1M [5] consists of 1M images each represented as a 128-dimensional SIFT descriptors [13]. It contains three vector subsets: learning set (100K), database set (1M) and query set (10K). The learning subset is retrieved from Flickr images and the database and query subsets are extracted from the INRIA Holidays images [4].

As stated in Section 3.3, our methods can be applied to different kinds of binary hashing methods. In our experiments, some representative hashing methods, Locality Sensitive Hashing (LSH) [1], PCA Hashing (PCAH) [20], Iterative Quantization (ITQ) [3], Spectral Hashing (SPH) [21] and Anchor Graph Hashing (AGH) [12], are chosen to evaluate the effectiveness of WhRank. The source codes generously provided by the authors and the recommended parameters settings in their papers, are used in our experiments. For AGH, the number of anchors is set to 500 and the number of nearest neighbors for anchor graph constructing is set to 2 for MINST70K and 5 for ANN-SIFT1M, respectively. Note that, the hash functions of LSH, PCAH and ITQ are linear, while those of SPH and AGH are nonlinear. Experimental results in Section 5.2 show that, WhRank is applicable to both linear and nonlinear hashing methods. Moreover, we also compare our algorithm with QsRank [22], a novel ranking algorithm for binary code. Since QsRank is developed only for PCA-based hashing methods, the comparisons are carried out on PCAH and ITQ.

Given a query, by ranking with traditional Hamming distance and our weighted Hamming distance, the returned top N nearest neighbors and the rankings are both different. The efficacy of WhRank can be measured by the Precision@N, Recall@N and the distance error ratio@N [15] defined as:

$$\begin{aligned}\text{Precision@}N &= \frac{\text{the number of similar points in top } N}{N} \\ \text{Recall@}N &= \frac{\text{the number of similar points in top } N}{\text{the number of all similar points}}\end{aligned}$$

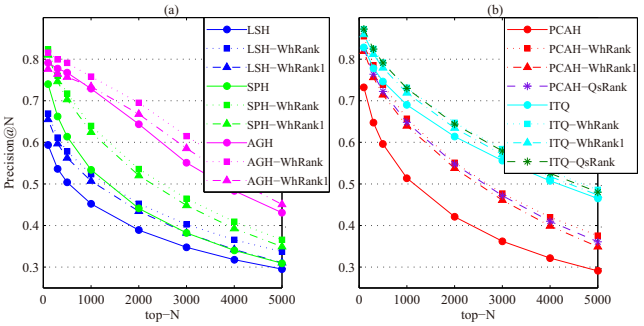

Figure 4. Evaluations of Precision@N of WhRank, WhRank1 and QsRank on MINST70K using 32 bits binary code. As shown in this figure, by applying WhRank for query results ranking, the retrieve accuracy of each method is improved.

$$\text{error ratio@}N = \frac{1}{N|Q|}\sum_{\boldsymbol{q}\in Q}\sum_{k=1}^{N}\frac{d(\boldsymbol{q},\boldsymbol{n}_k) - d(\boldsymbol{q},\boldsymbol{n}_k^*)}{d(\boldsymbol{q},\boldsymbol{n}_k^*)}$$

where $\boldsymbol{q} \in Q$ is a query, $\boldsymbol{n}_k$ is the k-th nearest neighbor in the ranked results, and $\boldsymbol{n}_k^*$ is the actual k-th nearest neighbor of $\boldsymbol{q}$ in the database set. For MINST70K, a returned point is considered as a true neighbor of a query if they share the same digit label. For ANN-SIFT1M, we use the same criterion as in [19]: a returned point is considered to be a true neighbor if it lies in the top 1% points closest to the query in terms of Euclidean distance in the original space.

5.2. Experimental Results

To demonstrate the efficacy of applying our weighted Hamming distance for ranking, given a query, the returned results of each baseline hashing method are ranked by their traditional Hamming distance and the weighted Hamming distance to the query respectively. The Precision@N and distance error ratio@N of each ranked result list are reported to show the efficacy of WhRank (the number of returned results is predefined in our experiments, a higher Precision@N means a higher Recall@N, thus only the Precision@N is reported.). Note that, ranking with the weighted Hamming distance is only performed to the results returned by computing the traditional Hamming distance, so the additional computational cost is minor.

Since MINST70K is fully annotated, we can use the Precision@N and Recall@N to show the efficacy of WhRank. The dataset is first embedded into Hamming space using each baseline hashing method. After that, from each digit class, we randomly sample 50 images from the query set constituting a subset contains 500 images. For each training image, we find its 1,000 neighbors in the dataset based on their digit labels. The training set and the corresponding neighbors are used for distribution parameters estimation. The rest of the query set is used as queries in our experiments. For LSH, PCAH and ITQ, the Gaussian distribution is used as the distribution assumption, while for SPH and AGH, the Laplace distribution is used as the distribution assumption.

Fig. 4 gives the Precision@N on MINST70K using 32 bits binary code. For clarity, the results are shown in two parts. It is easy to find out that, by ranking with our weighted Hamming distance (WhRank), all baseline hashing methods achieve a better search performance. On average, we get a 5% higher precision for each hashing method. For SPH and PCAH, the improvements are even higher (almost 10%). Meanwhile, as shown in this figure, each baseline method combined with WhRank1 also achieves a reasonable good performance improvement, and the improvement is a little inferior than that of WhRank (2% on average). In our subsequent experiments, this result still holds. Therefore, the results of WhRank1 are not given in subsequent figures for the sake of clarity.

Fig. 5 gives the Precision@N on MINST70K under different code lengths. Once again, we can easily find out that the performance of each baseline hashing method is improved when combined with WhRank. Moreover, as can be seen from Fig. 4 and Fig. 5(c), even with a relatively short binary code (32 bits), the retrieval accuracy of each baseline method combined with WhRank is almost the same as, sometimes better than, that of the baseline method itself with a binary code of larger size (64 bits, 96 bits).

In the experiments on ANN-SIFT1M, for distribution parameters estimation, we randomly sample 100 points from the query set as the training set, and for each training sample, we find its top 5,000 nearest neighbors in the database set, measured by the Euclidean distance. For LSH, PCAH and ITQ, we still use the Gaussian distribution as the distribution assumption. For SPH, the Laplace distribution is used as the distribution assumption. For AGH, the Student's t-distribution is used as the distribution assumption.

Since the neighborhood relationship of a data pair in ANN-SIFT1M is defined based on the Euclidean distance, we use Precision@N and distance error ratio@N to show the efficacy of ranking with our weighted Hamming distance. Fig. 6 and Fig. 7 give the evaluations of Recall@N and Distance error ratio@N on ANN-SIFT1M under different code lengths, respectively. As shown in these two figures, when combined with WhRank, each methods achieves a 10% higher precision on average. Moreover, the distance error ratio of each baseline method reduces 40% as compared with the original. The experimental results demonstrate that applying WhRank to existing hashing methods yielding a more accurate similarity search result.

We also compare our algorithm with QsRank [22]. Since QsRank is developed only for PCA-based hashing method, The comparisons are carried out on PCAH [20] and ITQ [3]. As QsRank is designed for ϵ-neighbor search, in our experiments on MINST70K, given a query $\boldsymbol{q}$ and N, the

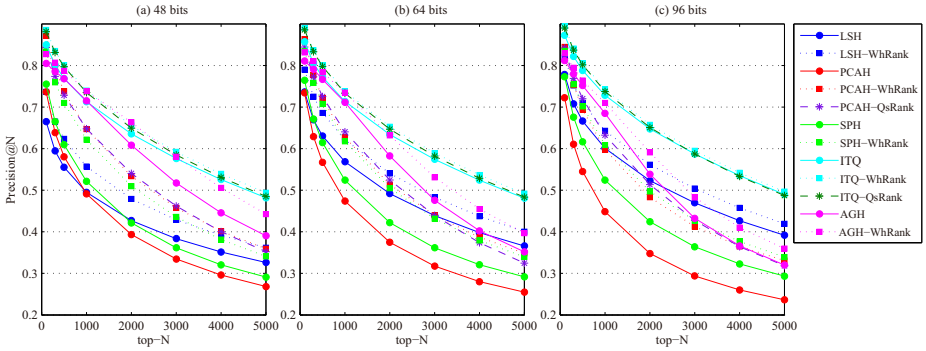

Figure 5. Evaluations of Precision@N of WhRank and QsRank on MINST70K. Code lengths: (a) 48 bits; (b) 64 bits; (c) 96 bits. As shown, the retrieve accuracy of each baseline method is improved when combined with WhRank under different code lengths.

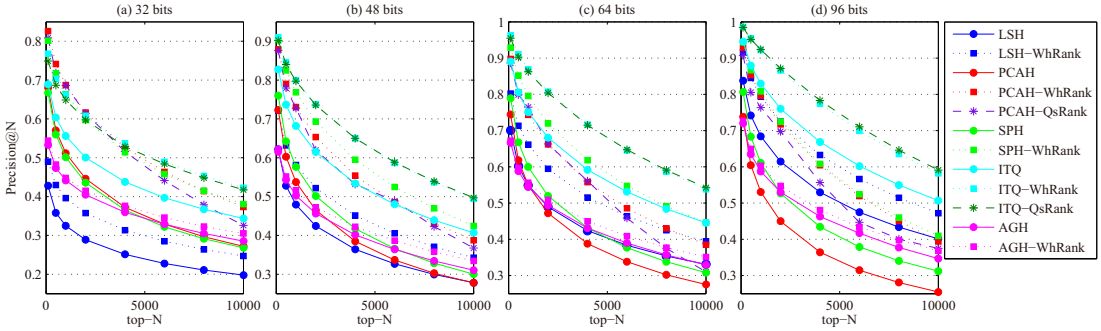

Figure 6. Evaluations of Precision@N of WhRank and QsRank on ANN-SIFT1M. Code lengths: (a) 32; (b) 48; (c) 64; (d) 96. The retrieve accuracy of each baseline method is improved when combined with WhRank under each code length setting. Moreover, the retrieve accuracy of each method combined with WhRank is as good as, sometimes better than, that of combined with QsRank.

search radius is set to the mean of the distance between $\boldsymbol{q}$ and its all neighbors. On ANN-SIFT1M, the radius is set to the distance between $\boldsymbol{q}$ and its actual N-th nearest neighbor in the database set. The comparison results are reported in Fig. 4(b) to Fig. 7. As shown in these figures, the performance improvements of our algorithm are as good as, sometimes better than, those of QsRank. One remarkable advantage of WhRank over QsRank is that, the ranking model of WhRank is more general, thus WhRank is also applicable to other non-PCA-based hashing methods, *e.g*. SPH and AGH. Furthermore, WhRank can be easily applied to ϵ-neighbor search, while QsRank is not very effective for nearest neighbor search since the distance between a query and its nearest neighbor is often unknown in practice.

6. Conclusion

Most existing binary hashing methods rank the returned results of a query simply with the traditional Hamming distance, which poses a critical issue for similarity search where ranking is important, since there can be many results sharing the same Hamming distance to the query. This paper proposes a weighted Hamming distance ranking algorithm (WhRank) to alleviate this ranking ambiguity. When applied to existing hashing methods, different bit-level weights are assigned to different hash bits, and the returned results can be ranked at a finer-grained binary code level rather than at the original integer Hamming distance level. We demonstrate that an effective bit-level weight is not only data-dependent but also query-dependent, and give a simple yet effective algorithm to learn the weights.

The experimental results on two large-scale image datasets containing up to one million high-dimensional data points demonstrate the efficacy of WhRank. The search performances of all evaluated hashing methods are improved when combined with WhRank. Moreover, as compared with QsRank, a novel ranking algorithm for binary code, the performance improvements of WhRank are as good as (sometimes better than) those of QsRank. There are two remarkable advantages of WhRank over QsRank. First, WhRank can be applied to various kinds of hashing methods while QsRank is only developed for PCA-based hashing methods. Second, as QsRank is developed for ϵ-neighbor search, it's not very effective for nearest neighbor search since the distance of a query to its nearest neighbor is unknown in practice. On the contrary, WhRank can be easily applied to ϵ-neighbor search.

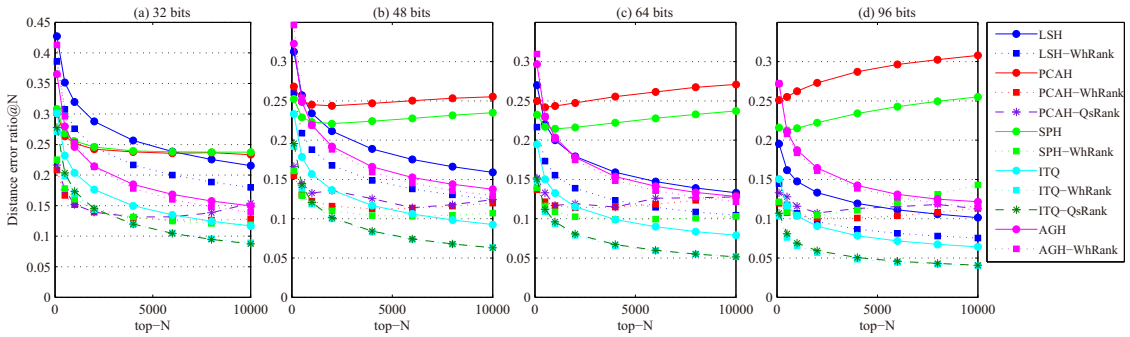

Figure 7. Evaluations of Distance error ratio@N of WhRank and QsRank on MINST70K and ANN-SIFT1M. Code lengths: (a) 32 bits; (b) 48 bits; (c) 64 bits; (d) 96 bits.

Acknowledgment

This work is supported by National Nature Science Foundation of China (61273247, 61271428), National Key Technology Research and Development Program of China (2012BAH39B02). This work was supported in part to Dr. Qi Tian by ARO grant W911BF-12-1-0057, NSF IIS 1052851, Faculty Research Awards by Google, FXPAL, and NEC Laboratories of America, and UTSA START-R Research Award (2012) respectively. This work was supported in part by NSFC 61128007 and NSFC under grant 61103059. This work was supported by the Importation and Development of High-Caliber Talents Project of Beijing Municipal Institutions (IDHT20130225), China Special Fund for Meteorological-scientific Research in the Public Interest (GYHY201106044) and NSFC 61271435.

References

[1] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In *IEEE Symp. Found. Comput. Sci.*, pages 459–468, 2006.

[2] J. L. Bentley and B. Labo. K-d trees for semidynamic point sets. In *Symp. Comput. Geom.*, pages 187–197, 1990.

[3] Y. Gong and S. Lazebnik. Iterative quantization: a procrustean approach to learning binary codes. In *Computer Vision and Pattern Recognition*, pages 817–824, 2011.

[4] H. Jégou, M. Douze, and C. Schmid. Improving bag-of-features for large scale image search. *International Journal of Computer Vision*, 87:316–336, 2010.

[5] H. Jégou, M. Douze, and C. Schmid. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:117–128, 2011.

[6] J. Ji, J. Li, S. Yan, B. Zhang, and Q. Tian. Super-bit locality-sensitive hashing. In *NIPS*, pages 108–116, 2012.

[7] Y. Jiang, J. Wang, and S.-F. Chang. Lost in binarization: query-adaptive ranking for similar image search with compact codes. In *ICMR*, page 16, 2011.

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

[9] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *International Conference on Computer Vision*, pages 2130–2137, 2009.

[10] Y. Lecun, L. E. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of The IEEE*, 86:2278–2324, 1998.

[11] W. Liu, J. Wang, R. Ji, Y. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *CVPR*, pages 2074–2081, 2012.

[12] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *ICML*, pages 1–8, 2011.

[13] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *IJCV*, 60:91–110, 2004.

[14] K. Min, L. Yang, J. Wright, L. Wu, X.-S. Hua, and Y. Ma. Compact projection: simple and efficient near neighbor search with practical memory requirements. In *Computer Vision and Pattern Recognition*, pages 3477–3484, 2010.

[15] L. Qin, W. Josephson, Z. Wang, M. Charikar, and K. Li. MultiProbe LSH: Efficient Indexing for High-Dimensional Similarity Search. In *VLDB*, pages 950–961, 2007.

[16] M. Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. In *Neural Information Processing Systems*, pages 1509–1517, 2009.

[17] R. Salakhutdinov and G. E. Hinton. Semantic hashing. *Int. J. Approx. Reason.*, 50:969–978, 2009.

[18] C. Silpa-anan and R. Hartley. Optimised KD-trees for fast image descriptor matching. In *CVPR*, pages 1–8, 2008.

[19] J. Wang, S. Kumar, and S.-F. Chang. Sequential Projection Learning for Hashing with Compact Codes. In *International Conference on Machine Learning*, pages 1127–1134, 2010.

[20] X. Wang, L. Zhang, F. Jing, and W. Ma. AnnoSearch: Image auto-annotation by search. In *CVPR*, 2006.

[21] Y. Weiss, A. B. Torralba, and R. Fergus. Spectral hashing. In *NIPS*, pages 1753–1760, 2008.

[22] L. Zhang, X. Zhang, and H.-Y. Shum. QsRank: Query-sensitive hash code ranking for efficient ϵ-neighbor search. In *CVPR*, pages 2058–2065, 2012.

[23] W. Zhou, Y. Lu, H. Li, Y. Song, and Q. Tian. Spatial coding for large scale partial-duplicate web image search. In *ACM Multimedia*, pages 511–520, 2010.

[24] W. Zhou, Y. Lu, H. Li, and Q. Tian. Scalar quantization for large scale image search. In *ACM Multimedia*, pages 169–178, 2012.