Distribution-Aware Locality Sensitive Hashing

Lei Zhang[1,2], Yongdong Zhang[1], Dongming Zhang[1], and Qi Tian[3]

[1] Advanced Computing Research Laboratory, Beijing Key Laboratory of Mobile Computing and Pervasive Device, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
[2] University of Chinese Academy of Sciences, Beijing, China
[3] University of Texas at San Antonio, San Antonio, USA
{zhanglei09,zhyd,dmzhang}@ict.ac.cn, qitian@cs.utsa.edu

Abstract. Locality Sensitive Hashing (LSH) has been popularly used in content-based search systems. There exist two main categories of LSH methods: one is to index the original data in an effective way to accelerate search process; the other one is to embed the high-dimensional data into hamming space and perform bit-wise operations to search similar objects. In this paper, we propose a new LSH scheme, called Distribution-Aware LSH (DALSH), to address the problem of lacking adaptation to real data, which is the intrinsic limitation of most LSH methods belong to the former category. In DALSH, a given dataset is embedded into a low-dimensional space with projection vectors learned from data, followed by deriving hash functions from the distribution of the dimension-reduced data. We also present a multi-probe strategy to improve the query performance. Experimental comparisons with the state-of-the-art LSH methods on two high-dimensional datasets demonstrate the efficacy of DALSH.

Keywords: Similarity search, Approximate nearest neighbor search, Locality sensitive hashing, Distribution-aware hashing.

1 Introduction

High-dimensional similarity search is a fundamental problem in many content-based search systems. To solve this problem efficiently, many index methods have been proposed, such as KD-tree [1] and R-tree [2]. These methods work well for low-dimensional data, whereas do not scale well with dimensionality because of "the curse of dimensionality" [3]. When the dimensionality exceeds 10, these methods are even inferior to linear-scan [4]. Fortunately, it is sufficient to find Approximate Nearest Neighbors (ANN) in many applications [5]. Many methods for ANN search are developed and among these methods, Locality Sensitive Hashing (LSH) [6, 7] and its variants [8, 9] are well-known for their efficacy. LSH was first proposed by Indyk et al. in [6]. It has been extended to a variety of similarity metrics beyond Euclidean distance [7, 20], including hamming distance [5], p-norm distance [7] and kernel similarity [10, 21]. In general, there exist two main categories of LSH methods: one is to index the original data in an effective way to accelerate the search process (we call "original" LSH); the other one is to embed the high-dimensional data into hamming space and perform bit-wise operations to search similar objects (called binary LSH).

S. Li et al. (Eds.): MMM 2013, Part II, LNCS 7733, pp. 395–406, 2013.

A famous "original" LSH method is the Euclidean LSH (E2LSH) [7], which uses linear projections to partition a given dataset into several subsets. Some variations [8, 12] based on multi-probe strategy have been brought forward to reduce the storage space. In [13], LSH Forest is proposed to eliminating the different data-dependent parameters for which LSH must be constantly hand-tuned. Several other hash methods have been proposed to improve the query performance, such as Leech lattices [14] and E8 lattices [15]. In lattice-based hashing, any two points in the same lattice are separated by a bound distance which only depends on the lattice definition. The lattice-based hash functions exploit the vectorial structure of Euclidean space, thus outperform E2LSH. In [16], Paulevé et al. use the k-means clustering to partition a dataset and propose k-means LSH (KLSH). A given dataset is first clustered into several subsets, and then for each subset, its cluster center is used as the identifier. Given a query, only the subset with the nearest center to the query is searched. It is argued that KLSH is the most effective algorithm of LSH [16]. To our knowledge, it is the most effective LSH method which belongs to the "original" LSH methods.

The other kind of LSH methods, binary LSH, and more generally, binary hashing, is becoming increasingly popular for efficient nearest neighbor search. Given a dataset, binary hashing generates binary code for each object and approximates the distance or similarity of two objects by the hamming distance between their binary codes. To generate compact binary code, many algorithms have been proposed. In semantic hashing [17], the Restricted Boltzmann Machine is used to map similar objects to similar codes. Spectral hashing [18] utilizes the simple analytical eigenfunction solution of 1D Laplacians as binary hash function. Shift-Invariant Kernel Hashing, a distribution-free method based on the random features mapping for shift-invariant kernels, is proposed in [19] and has a comparable performance to that of Spectral Hashing. Moreover, to exploit the spectral properties of the data affinity (e.g. pairwise similarity) to generate better binary codes, many other algorithms, such as Semi-supervised Sequential Projection Hashing [9], Anchor Graph Hashing [11] and Kernel-Based Supervised Hashing [21], have been developed and give commendable search performance. These methods are generally superior to LSH because LSH is simple, always data-independent and has no learning algorithm to reveal the underlying information of dataset. Inspired by these research works, we introduce a supervised learning algorithm to "original" LSH methods to address the problem of lacking adaptation to real data within these methods.

Hash functions of LSH determine the way a dataset indexed, hence, they are the core elements that affect the query performance. In E2LSH, the hash functions are quantized linear projections with randomly selected projection vectors. Since each projection vector is randomly selected, E2LSH partitions a given dataset without taking account of the dataset information. Therefore, the final dataset partition is not optimal and this brings some limitations. A main limitation is that multi hash tables are needed to guarantee the query accuracy. In lattice-based hashing [14, 15], the hash functions only exploit the vectorial structure of Euclidean space, whereas the hidden information of the dataset is still discarded. To make the constructed index data-adaptive, Paulevé et al. [16] have proposed k-means based LSH (KLSH). In KLSH, a given dataset is clustered into k subsets, then for each subset, the cluster center is

used as the identifier. Given a query q, only the cluster with the nearest center to q is searched. To reduce the complexity of the clustering stage, the hierarchical k-means (HKM) [16] is used. This method consists of a series of k-means with a relatively small k and produces a balanced tree structure as follows: a dataset is first clustered into k (branching factor $b_f = k$) subsets, then each subset is also clustered into k sub-subsets using k-means. This process proceeds recursively until obtaining a pre-defined tree height h_t.

Since the k-means clustering is performed in the original data space and it minimizes the inner class distances, the final dataset partition adapts to the data distribution well. It is argued that KLSH is the most effective algorithm of LSH [16]. However, k-means clustering only ensures to find local minimum, especially in high-dimensional space, hence the dataset partition is non-optimal when the dimensionality is high. Moreover, since a high-dimensional vector is used as the identifier of each subset, the memory occupation of KLSH is relatively larger than that of E2LSH.

In this paper, we propose a new LSH scheme called Distribution-Aware LSH (DALSH). It alleviates the problem of lacking adaptation to real data, which is the intrinsic limitation of most existing "original" LSH methods. Linear projection, with projection vectors learned by a supervised learning algorithm, is used to reduce the dimension of the dataset to index. Then, the hash functions are derived from the distribution of the dimension-reduced data. Experimental results on two public high-dimensional datasets demonstrate the efficacy of the proposed method as compared with the state-of-the-art "original" LSH methods, Kmeans LSH. The rest of this paper is organized as follows. In Section 2, we present our DALSH algorithm and give the multi-probe strategy to improve the query performance. Section 3 describes our experiments and Section 4 concludes this paper.

2 Distribution-Aware Hashing

2.1 Supervised Projection Learning

The hash functions of DALSH are derived from the distribution of the dataset, hence, the dataset distribution is essential. There are several novel methods that can construct predictive models for high-dimensional data with high accuracy. However, most of them are computationally expensive. To simplify the process of distribution modeling, we first use linear projection to embed the original data into a low-dimensional space. The projection vectors are learned from the data with a supervised learning algorithm.

Given a dataset $\boldsymbol{X} = [x_1, x_2, \dots, x_n] \in R^{d \times n}$ with a fraction of points associated with two categories of relationships: $\mathcal{N}$ and $\mathcal{C}$. A tuple $(x_i, x_j) \in \mathcal{N}$ is denoted as a neighbor pair, while $(x_i, x_j) \in \mathcal{C}$ is denoted as a non-neighbor pair. Suppose there are l $(l < n)$ points, each of which is associated with at least one of these two categories. The matrix formed by these l data points is denoted as $\boldsymbol{X}_l \in R^{d \times l}$. For a projection vector ω, we want the difference between the projections of a neighbor pair on it small while the difference between the projections of a non-neighbor pair large. As a result, the empirical accuracy of ω can be measured as follows:

$$J(\omega) = \frac{1}{2}\left(\sum_{(x_i,x_j)\in\mathcal{N}} \left|\omega^T(x_i - x_j)\right|^2 sim(x_i,x_j) - \sum_{(x_i,x_j)\in\mathcal{C}} \left|\omega^T(x_i - x_j)\right|^2 sim(x_i,x_j) \right) . \quad (1)$$

where $sim(x_i,x_j) = \exp\left\{-\left\|x_i - x_j\right\|_2^2/\sigma^2\right\}$[18]. Minimizing $J(\omega)$ would make the difference between projections of a neighbor pair small; the difference between projections of a non-neighbor pair large. With simple algebra, (1) can be rewritten as:

$$J(\omega) = \omega^T \boldsymbol{X}_l \boldsymbol{L} \boldsymbol{X}_l^T \omega \ . \quad (2)$$

where $\boldsymbol{L} = \boldsymbol{D} - \boldsymbol{S}$, $\boldsymbol{S} \in R^{l\times l}$ is a label matrix defined in (3) and $\boldsymbol{D} = diag\{\boldsymbol{S}\boldsymbol{1}\}$.

$$\boldsymbol{S}_{ij} = \begin{cases} sim(x_i,x_j)\text{: } (x_i,x_j) \in \mathcal{N} \\ -sim(x_i,x_j)\text{: } (x_i,x_j) \in \mathcal{C} \\ 0\text{: } otherwise \end{cases} . \quad (3)$$

Let $\boldsymbol{W} = [\omega_1, \omega_2, \dots, \omega_K] \in R^{d\times K}$, where $\omega_1, \omega_2, \dots, \omega_K$ are projection vectors. These K projection vectors are solutions of the following optimization problem:

$$\boldsymbol{W} = \arg\min_{\omega_k\in R^d} \sum\nolimits_k \omega_k^T \boldsymbol{X}_l \boldsymbol{L} \boldsymbol{X}_l^T \omega_k = \arg\min_{\boldsymbol{W}\in R^{d\times K}} tr\{\boldsymbol{W}^T \boldsymbol{X}_l \boldsymbol{L} \boldsymbol{X}_l^T \boldsymbol{W}\} \ . \quad (4)$$

(4) can be easily solved if the orthogonality constraints are imposed on ω_k (i.e., $\boldsymbol{W}^T\boldsymbol{W} = \boldsymbol{I}$). The solutions are simply the eigenvectors with the top K smallest eigenvalues of $\boldsymbol{X}_l \boldsymbol{L} \boldsymbol{X}_l^T$. However, the orthogonality constraint is not always reasonable and relaxing this constraints may lead to a better result [9], thus, we add a perturbation matrix $\boldsymbol{\epsilon} \in R^{d\times K}, \epsilon_{ij} \sim \mathcal{N}(0,\sigma_\epsilon)$ and set $\boldsymbol{W} = \boldsymbol{W_{opt}} + \boldsymbol{\epsilon}$.

2.2 Distribution-Aware Hashing Function

The hash value is obtained by partitioning the data space into several subspaces with several hyperplanes perpendicular to ω_k. In this way, each data point belongs to a unique subspace and the identifier of the subspace is used as the hash value. In Section 2.1, only the labeled data are used to learn ω_k, this may lead to overfitting. To alleviate this potential problem, the information entropy of hash function, which utilizes both labeled and unlabeled data, is used as the regularizer. Denote the hash function associated with ω_k as $h_k(x)$, the entropy of $h_k(x)$ is:

$$H(h_k(x)) = \sum\nolimits_z - \Pr(h_k(x) = z) \log \Pr(h_k(x) = z) \ . \quad (5)$$

To maximize $H(h_k(x))$, $\Pr(h_k(x) = z)$ should be identical for all hash values z. Denote CDF_k the cumulative distribution function of projections on ω_k, we have:

$$h_k(x) = \lfloor L_k \cdot CDF_k(\omega_k^T x) \rfloor \ . \tag{6}$$

Each $h_k \in H: R^d \to Z$ maps a $x \in R^d$ to an integer $z \in \{0, \ldots, L_k - 1\}$. It maps x to a line, and then the line is divided into L_k slots. The hash value of DALSH is a concatenation of K hash values: $g(x) = \left(h_1(x), h_2(x), \ldots, h_K(x)\right)^T$.

It has been reported in the literature that the distribution of projections of high-dimensional data on a randomly selected direction follows a Gaussian distribution [22]. Therefore, we use the Gaussian mixture model (GMM) with EM algorithm to estimate the distribution. The probability density function (PDF) of projections on ω_k is a mixture of Gaussian distributions: $\mathrm{PDF_k} = \sum_{i=1}^{g_k} \pi_{ki} \mathcal{N}(\mu_{ki}, \sigma_{ki})$, where g_k is the number of Gaussian components in PDF_k.

There is a hidden problem of this hashing strategy. Since the length of slot is relatively small in dense areas, two points p, q with distance $r = \|p - q\|_2$ in a dense area are less likely to map to the same slot than two other points with the same r but in a sparse area. However, the experimental results indicate that this problem does not affect the query performance greatly. Moreover, the multi-probe strategy proposed in Section 2.3 can alleviate this hidden problem efficiently.

2.3 Multi-probe Strategy

A hash *perturbation vector* is defined as: $\Delta = (\delta_1, \delta_2, \ldots, \delta_K)^T$. Given a query q, not only the bucket $g(q)$, but also the buckets $g(q) + \Delta$ that are likely to contain similar objects of q, are searched. Given the locality-sensitive of hash function (6), if an object similar to q is not in $g(q)$, then it is likely in the bucket with hash value slightly different from $g(q)$. Fig. 1 shows the distribution of bucket distances of *N* nearest neighbors. The left of Fig. 1 shows that for a query, more than 90% of the single hash values of its top-*N* nearest neighbors are either same as that of the query or differ by just -1 or 1, which indicates a majority of $\delta_k \in \{0, -1, 1\}$. The right of Fig. 1 shows that the probabilities of $\delta_k = 0$ are almost the same at each dimension. Hence, we restrict our attention to Δ with $\delta_k \in \{-1, 0, 1\}$. For a query q, its perturbation vectors can be derived based on the probability of finding an similar object of q in bucket $g(q) + \Delta$(denoted by $\Pr(\Delta)$). Denote $\Pr_k(\delta_k)$ the probability of $(g(q) + \Delta)_k$ differs by δ_k from $g(q)_k$ and assume all the hash dimensions are independent, we have:

$$\Pr(\Delta) = \prod_{k=1}^{K} \Pr_k(\delta_k), \delta_k \in \{-1, 0, 1\} \ . \tag{7}$$

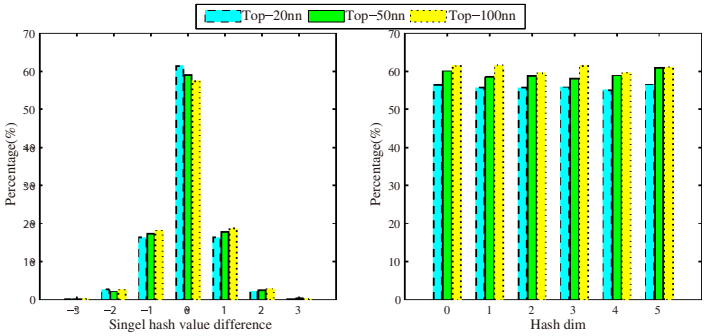

Fig. 1. Distribution of the bucket distances. The dataset used for illustration is SIFT1M [23], hash dim $K = 6$. The left is the distribution of single hash value differences (δ_k), the right is the probability of each hash dimension remains unchanged in perturbation vectors.

Normalize each $\Pr(\Delta)$ by dividing $\Pr(\mathbf{0}) = \prod_k \Pr_k(0)$ and take the negative logarithm of $\Pr(\Delta) / \Pr(\mathbf{0})$, (7) can be rewritten as:

$$log\_score(\Delta) = \sum_{k=1,\delta_k \neq 0}^{K} -\log \Pr'_k(\delta_k) \ . \tag{8}$$

where $\Pr'_k(\delta_k) = \Pr_k(\delta_k)/\Pr_k(0)$, $\Pr'_k(-1) + \Pr'_k(1) = 1/\Pr_k(0) - 1$.

$log\_score(\Delta)$ only depends on the non-zero dimensions of Δ. Therefore, without taking account of $\Pr_k(0)$ as [8] dose, we define a score function to measure the probability of finding nearest neighbor of q in $g(q) + \Delta$:

$$score(\Delta) = \sum_{k=1,\delta_k \neq 0}^{K} -\log\big(p_k(\delta_k)\big) \ . \tag{9}$$

where $p_k(\delta_k) \propto \Pr'_k(\delta_k) \propto \Pr_k(\delta_k)$. In $score(\Delta)$, only $p_k(\delta_k)$ with $\delta_k \neq 0$ are summed, which means in $g(q) + \Delta$, only the dimensions different from $g(q)$ make contribution to $score(\Delta)$. Hence, $p_k(\delta_k)$ can be considered as a "generative probability", and in this case we can set $p_k(-1) + p_k(1) = 1$. In the following, we show how to get a proper estimation of $p_k(\delta_k)$ with a simple method.

Fig. 2 illustrates the probability of q's nearest neighbors falling into the neighboring slots. Here $f_k(q) = \omega_k^T q$, $h_k(q)$ is the slot to which q maps. W_{ku}, W_{kl} are the upper and lower boundaries of slot $h_k(q)$ respectively. Let $x_k(-1) = \mathrm{CDF}_k\big(f_k(q)\big) - \mathrm{CDF}_k(W_{kl})$, $x_k(1) = \mathrm{CDF}_k(W_{ku}) - \mathrm{CDF}_k\big(f_k(q)\big)$. Obviously, the larger $x_k(-1)$ is, the larger $\Pr_k(1)$ is; the lager $x_k(1)$ is, the larger $\Pr_k(-1)$ is. Since $\Pr_k(-1) + \Pr_k(1)$ and $x_k(-1) + x_k(1)$ (which is $1/L$, L is the number of slots) are both constants, a reasonable assumption of the linear relationship between $x_k(\delta_k)$ and $\Pr_k(\delta_k)$ is: $\Pr_k(1) \propto x_k(-1)$; $\Pr_k(-1) \propto x_k(1)$. Moreover, we also have $p_k(-1) + p_k(1) = 1$ and $p_k(\delta_k) \propto \Pr_k(\delta_k)$. Therefore, based on the above derivations and approximations, we get a reasonable estimation of $p_k(\delta_k)$:

$$p_k(-1) = \frac{x_k(1)}{x_k(1) + x_k(-1)}, \quad p_k(1) = \frac{x_k(-1)}{x_k(1) + x_k(-1)} \ . \tag{10}$$

Given a query q, its corresponding $p_k(-1)$ and $p_k(1)$ for each dimension k is:

$$p_k(-1) = h_k(q) + 1 - L \cdot \mathrm{CDF}_k(\omega_k^T q) \ , \ \ p_k(1) = 1 - p_k(-1) \ . \tag{11}$$

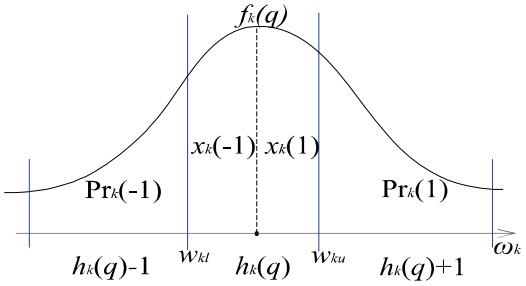

Fig. 2. Illustration of the probability of q's nearest neighbors falling into the neighboring slots

The mathematical form of $score(\Delta)$ (9) is exactly the same as the score function defined in [8], hence, we could use the algorithms proposed in [8] to derive the perturbation vectors. After deriving perturbation vectors Δ, buckets $g(q) + \Delta$ are searched according to the ascending order of $score(\Delta)$. The complexity of deriving T perturbation vectors is almost $O(KlogK + \sum_{i=1}^{T} 2\log i) < O(KlogK + T\log T)$. Note that, some approximations are made to simplify the derivation of Δ, and this may cause the final perturbation vectors non-optimal. However, the experimental results in Section 3 demonstrate the effectiveness of our multi-probe strategy, which indicate the efficacy of the derived perturbation vectors.

3 Experimental Result

Two public high-dimensional datasets [23] are used in our experiments, one consists of 1M 128-dimensional local SITF features [24] (SIFT1M) and the other one consists of 1M 960-dimensional global GIST features [25] (GIST1M). Each dataset contains three vector subsets (database, query and learning) and a groundtruth set. For SIFT1M, the data points of each vector subset are normalized by dividing each dimension by the largest l_2 norm in the database set.

3.1 Performance Metric

Search accuracy and *Search complexity* are used to measure the similarity search performance of our method. These two measures reflect the objective which is of interest in many practical applications.

Search Accuracy. For ANN query, *Recall* (12) is used as the performance metric. It is the percentage of queries that the true nearest neighbor is in the result list [16].

$$\text{Recall} = \frac{1}{|Q|}\sum_{q\in Q}\delta(q) \ . \tag{12}$$

$\delta(q)$ is 1 if q's nearest neighbor is in the result list or 0 if not. For k-Nearest Neighbor Search (kNN), distance *error ratio* [8] is used as the performance metric:

$$\text{error ratio} = \frac{1}{K|Q|}\sum_{q\in Q}\sum_{k=1}^{K}\frac{d_k - d_k^*}{d_k^*} \ . \tag{13}$$

where d_k is the distance between q and its *k*-th nearest neighbor in the result list, d_k^* is the distance between q and its true *k*-th nearest neighbor.

Search Complexity. The complexity of search process is of concern to many practical applications. Since the search process of our algorithm is similar to that of [16], the search complexity model in [16] is applicable to our algorithm. Two major phrases for search process are defined in [16] and listed below:

Phrase 1: query preparation cost (*qpc*). It measures the complexity of identifying the buckets that the search process will subsequently analyze in detail.

Phrase 2: short-list processing cost: *selectivity* (*sel*). It is the fraction of the number of data points in the result list to the number of data points in total dataset.

The overall search cost of our algorithm is: $\text{sel}\cdot nd + \text{qpc}$. The acceleration factor *ac* [16] defined in (14) is used to measure the speedup over linear search:

$$\text{ac} = \frac{nd}{\text{sel}\cdot nd + qpc} = \frac{1}{\text{sel} + qpc/(nd)} \ . \tag{14}$$

3.2 Experimental Setup

Label Matrix Construction. We randomly sample l points from the learning set to form the neighbor and non-neighbor pairs. The distance between each data point p in the sample set $\boldsymbol{X}_l$ and its 150th nearest neighbor is calculated and averaged ($\bar{d}_{150}$). For any $x_i, x_j \in \boldsymbol{X}_l$: if $d(x_i, x_j) \le \bar{d}_{150}$, (x_i, x_j) is a neighbor pair; if $d(x_i, x_j) \ge t\bar{d}_{150}$, (x_i, x_j) is a non-neighbor pair; otherwise, the relationship of x_i and x_j is not defined. For SIFT1M, l=2000, t=1.5, and for GIST1M, l=5000, t=2.5.

Parameters Settings. There are three key parameters influencing the performance of DALSH: the dimensionality of hash value K, the number of slots of each hash dimension L (L_k is set to the same value) and the number of Gaussian components g_k. K and L are varied in our experiments to get different performance. The KL divergence between the actual and estimated distributions is used to adjust g_k. With a larger g_k, the estimation is more accurate. However, the performance is not improved

much when g_k exceeds 4, moreover, a larger g_k may also lead to overfitting. Therefore, in our experiments, g_k is set to 3 in the experiments on dataset SIFT1M and 4 on dataset GIST1M. The experimental results are averaged over all queries (10,000 for SIFT1M and 1,000 for GIST1M) with one single hash table.

3.3 Comparisons with "original" LSH Methods

In [16], Paulevé et al. have evaluated several hash functions of LSH by comparing Recall (12) of each hash function at a given selectivity. They argue that the k-means based LSH (KLSH) gives the best result. Since recall at a given selectivity reflects the effectiveness of a hash function, we do the same evaluation for DALSH with one single hash table. Fig. 3 gives the evaluation of distribution-aware hashing (6) compared with several other hashing schemes evaluated in [16][1]. Note that, each point in Fig. 3 corresponds to an optimal parameters setting that gives the best performance for a given selectivity. As can be seen from this figure, the distribution-aware hashing significantly outperformers the random projection [8] and lattice [15] based hashing: the selectivity is almost one order of magnitude smaller for the same recall. Moreover, the performance of distribution-aware hashing is almost as good as (even better than) that of the k-means based hashing (k-means, HKM) when recall is larger than 40%.

We evaluate the performance of DALSH for similarity search compared with the popularly used E2LSH and the state-of-the-art "original" LSH method, Kmeans LSH (KLSH). We report the acceleration factor (14) of each method at the same Recall. In DALSH, the complexity of Phrase 1 is almost $O(KlogK + T\log T) + O(dK)$. Since $n, d \gg K$, $nd \gg T\log T$, the acceleration is almost $1/\mathrm{sel}$. The top row of Fig. 4 gives the experimental results for ANN query on SIFT1M and GIST1M respectively. As shown, our method gives the best performance on both two datasets: its recall is almost 10%-20% higher than those of E2LSH and KLSH, and its query speed is almost 6 and 1.2-4 times faster than those of E2LSH and KLSH respectively.

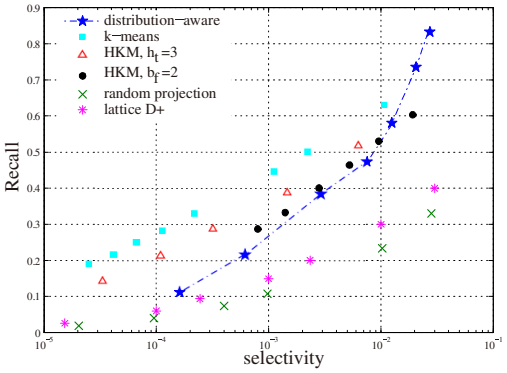

Fig. 3. Evaluation of the distribution-aware hash function on SIFT1M, compared with the (hierarchical) k-means, random projection and lattice based hash functions evaluated in [16]

[1] The dataset used in our experiment is the same as [16], thus the experimental results of us and [16] can be compared directly. Only some representative points of the "selectivity-recall" curve in [16] are depicted, for more details, please refer to Fig. 2 of [16].

For kNN query, the error ratio (13) is used as the performance metric and we set K=100. As shown in the bottom row of Fig. 4, DALSH and KLSH give almost the same performance (both outperform E2LSH), their query speed are both almost 10 times faster than that of E2LSH. It is not surprising that KLSH gives the best performance for kNN query, since the inner distance of each cluster in minimized when clustering. However, in this case, the acceleration of KLSH is relatively small.

3.4 More Comparisons

We also compare DALSH with several other kinds of high-dimensional index methods, such as optimized KD-tree [26], FALNN [27] and spectral hashing [18]. The experimental results are succinctly presented below due to the space limitation.

When compared with spectral hashing (SpH), the code length is set to 16, 32 and 64, and the query results are those with hamming distance to the query less than 6. SpH's performance is not improved much with a longer code and the highest recall is almost 40% on SIFT1M, which is inferior to that of DALSH.

The optimized KD-tree is implemented within FLANN. We change the parameter "algorithm" and "target_precision" to get different performance. In the experiments on SIFT1M, the best recall of optimized KD-tree is 67.35% with a speedup less than 100, both are inferior to those of DALSH. The best performance of FALNN is obtained by setting "algorithm" to "FLANN_INDEX_AUTOTUNED", and in this case, the recall is 80.45% and the speedup is 77.2, while the speedup of DALSH is almost 200 at the same recall. On GIST1M, since the dimension is high (960), FLANN's performance degrades greatly, which is consistent with the result in [27], while DALSH's performance is almost unchanged (see. Fig. 4).

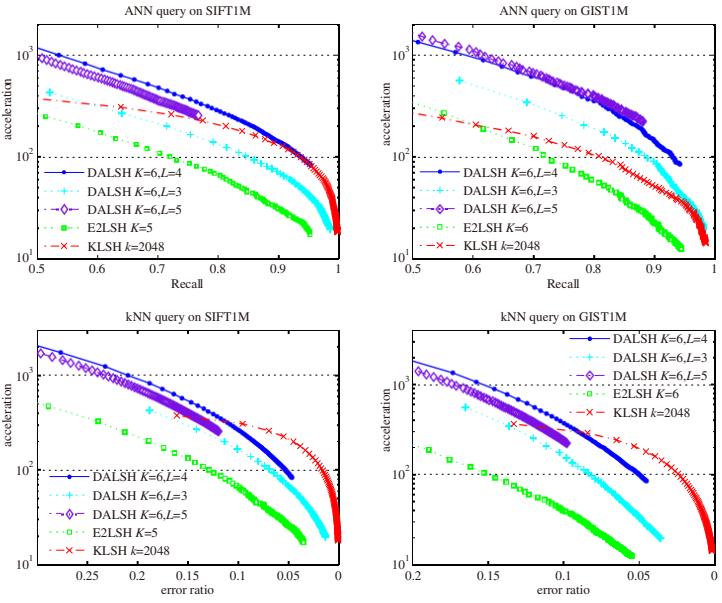

Fig. 4. Evaluations of DALSH, compared with E2LSH and KLSH. L is the number of slots in DALSH, K is the hash dimension, k is the number of clusters in KLSH. The top and bottom row give the acceleration of DALSH over linear search for ANN and kNN query respectively.

4 Conclusions

In this paper, we propose the Distribution-Aware LSH (DALSH) for high-dimensional similarity search. With a supervised learning algorithm, we generate a series of data-adaptive projection vectors, each of which tries to minimize the difference between the projections of similar objects, while maximize the difference between the projections of dissimilar objects on it. Linear projection is used to reduce the dimensionality of the dataset to index, and then the hash functions of DALSH are derived from the distribution of the dimension-reduced data. In this way, the problem of lacking adaptation to real data, which is the intrinsic limitation of most existing "original" LSH methods, is alleviated. In addition, we present an efficient multi-probe strategy to improve the query performance of DALSH. The experimental results on two public high-dimensional datasets demonstrate the efficacy of DALSH. Compared with the state-of-the-art "original" LSH method, Kmeans LSH (KLSH), DALSH shows commendable performance gains in terms of search accuracy and search efficiency. Moreover, compared with other kinds of index methods (e.g. optimized KD-tree, spectral hashing), DALSH still gives a better performance.

Acknowledgments. This work is supported by National Nature Science Foundation of China (61273247, 61271428), National Key Technology Research and Development Program of China (2012BAH39B02), and Co-building Program of Beijing Municipal Education.

References

1. Bentley, J.L.: K-d trees for semidynamic point sets. In: Proc. SCG, pp. 187–197 (1990)
2. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: International Conference on Management of Data, pp. 47–57 (1984)
3. Beyer, K., Goldstein, J., Ramakrishnan, R., Shaft, U.: When is "nearest neighbor" meaningful? In: International Conference on Database Theory, pp. 217–235 (1999)
4. Tao, Y., Yi, K., Sheng, C., Kalnis, P.: Efficient and accurate nearest neighbor and closest pair search in high-dimensional space. ACM TODS 35(3), 20 (2010)
5. Gionis, A., Indyk, P., Motwani, R.: Similarity search in high dimensions via hashing. In: Proceedings of the International Conference on Very Large Data Bases, pp. 518–529 (1999)
6. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proc. STOC, pp. 604–613 (1998)
7. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Proc. SCG, pp. 253–262 (2004)
8. Lv, Q., Josephson, W., Wang, Z., Charikar, M., Li, K.: Multi-probe lsh: efficient indexing for high-dimensional similarity search. In: Proc. VLDB, pp. 950–961 (2007)
9. Wang, J., Kumar, S., Chang, S.F.: Sequential projection learning for hashing with compact codes. In: Proc. ICML, pp. 1127–1134 (2010)
10. Kulis, B., Grauman, K.: Kernelized locality-sensitive hashing for scale image search. In: International Conference on Computer Vision, pp. 2130–2137 (2009)

11. Liu, W., Wang, J., Kumar, S., Chang, S.F.: Hashing with graphs. In: ICML, pp. 1–8 (2011)
12. Joly, A., Buisson, O.: A posteriori multi-probe locality sensitive hashing. In: ACM MM (2008)
13. Bawa, M., Condie, T., Ganesan, P.: LSH forest: self-tuning indexes for similarity search. In: Proc. WWW, pp. 651–660 (2005)
14. Shakhnarovich, G., Darrell, T., Indyk, P.: Nearest-neighbor methods in learning and vision. IEEE Transactions on Neural Networks 19(2), 337 (2008)
15. Jégou, H., Amsaleg, L., Schmid, C., Gros, P.: Query adaptative locality sensitive hashing. In: IEEE Int. Conf. on Acoustics, Speech and Signal Processing, pp. 825–828 (2008)
16. Paulevé, L., Jégou, H., Amsaleg, L.: Locality sensitive hashing: A comparison of hash function types and querying mechanisms. Pattern Recognition Letters 31(11), 1348 (2010)
17. Salakhutdinov, R., Hinton, G.: Semantic hashing. Int. J. Approx Reason. 50(7), 969 (2009)
18. Weiss, Y., Torralba, A., Fergus, R.: Spectral hashing. In: NIPS, pp. 1753–1760 (2008)
19. Raginsky, M., Lazebnik, S.: Locality-sensitive binary codes from shift-invariant kernels. In: Neural Information Processing Systems, pp. 1509–1517 (2009)
20. Wang, M., Yang, K., Hua, X., Zhang, H.: Towards a relevant and diverse search of social images. IEEE Transactions on Multimedia, 829–842 (2010)
21. Liu, W., Wang, J., Ji, R., Jiang, Y.G., Chang, S.F.: Supervised hashing with kernels. In: Proc. CVPR, pp. 2074–2081 (2012)
22. Lejsek, H., Ásmundsson, F.H., Jónsson, B.T., Amsaleg, L.: NV-tree: An efficient disk based index for approximate search in very large high-dimensional collections. IEEE Transactions on Pattern Analysis and Machine Intelligence 31(5), 869 (2009)
23. Jégou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Transactions on Pattern Analysis and Machine Intelligence 33(1), 117 (2011)
24. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 91–110 (2004)
25. Oliva, A., Torralba, A.: Modeling the shape of the scene: A holistic representation of the spatial envelope. International Journal of Computer Vision 42(3), 145 (2001)
26. Silpa-Anan, C., Hartley, R.: Optimised KD-trees for fast image descriptor matching. In: IEEE Conference on Computer Vision and Pattern Recognition, pp. 1–8 (2008)
27. Muja, M., Lowe, D.G.: Fast approximate nearest neighbors with automatic algorithm configuration. In: Int. Conf. Computer Vision Theory and Applications, pp. 331–340 (2009)