Compact Hashing with Joint Optimization of Search Accuracy and Time

Junfeng He
Columbia University
New York, New York, 10027, USA
jh2700@columbia.edu

Regunathan Radhakrishnan
Dolby Laboratories
San Francisco, CA, 94103, USA
regu.r@dolby.com

Shih-Fu Chang
Columbia University
New York, New York, 10027, USA
sfchang@ee.columbia.edu

Claus Bauer
Dolby Laboratories
San Francisco, CA, 94103, USA
cb@dolby.com

Abstract

Similarity search, namely, finding approximate nearest neighborhoods, is the core of many large scale machine learning or vision applications. Recently, many research results demonstrate that hashing with compact codes can achieve promising performance for large scale similarity search. However, most of the previous hashing methods with compact codes only model and optimize the search accuracy. Search time, which is an important factor for hashing in practice, is usually not addressed explicitly. In this paper, we develop a new scalable hashing algorithm with joint optimization of search accuracy and search time simultaneously. Our method generates compact hash codes for data of general formats with any similarity function. We evaluate our method using diverse data sets up to 1 million samples (e.g., web images). Our comprehensive results show the proposed method significantly outperforms several state-of-the-art hashing approaches.

1. Introduction and related work

The explosive growth of data on the Internet has given rise to many interesting applications, for many of which a core technical issue involves fast nearest neighbor search over a large data collection in response to a query. Brute-force solutions using linear scan obviously do not scale up when the data size grows due to the excessive cost in storage and processing. Recent works [1, 2, 3, 5, 6, 11, 10, 8, 13, 14, 12, 17], have reported interesting results in finding approximate solutions over large data sets. Early works in this field focused on finding appropriate spatial partitions of the feature space via various tree structures[1, 2]. With good performance for low dimensional data, such tree-based approaches however, are known to degrade significantly when the data dimension is high, sometimes even worse than linear scan. Recently, many hash coding based algorithms have been proposed to handle similarity search of high dimensional data [3, 5, 6, 11, 10, 8, 13, 14, 12, 17]. Up to now, there are two main groups of hashing methods: hashing based on random projections and hashing for compact codes.

Among the popular randomized hash based techniques is the locality-sensitive hashing (LSH) [3]. In LSH, random vectors (with some specific distribution) are used as the projection bases. As an important property of LSH, points with high feature similarity are proven with a high probability to obtain the same hash code, which guarantees a asymptotic theoretical property of sublinear search time. Variations of LSH algorithms have been proposed in recent literatures to expand its usage to the cases of inner products [4], L_p norms [5], and learned metrics [7], etc.

Despite of its success, one arguable disadvantage of LSH is the inefficiency of the hash codes. Since the hash functions in LSH are randomly generated and independent of the data, it is not very efficient. Usually, it needs long codes in each hash table to achieve an acceptable accuracy. However, since the collision probability decreases very fast (exponentially) with the length of the code, when long codes are used in one hash table, then many hash tables are often needed to get a good recall. This would heavily increase the requirement of storage, causing problems for very large scale applications. As demonstrated in [11, 10], if we want to scale up to Internet size databases (80 millions samples in their experiments), fitting those large scale data into a desktop memory, e.g., a few Gigabytes, means we can only use a small number of bits per sample. In this case, LSH with short codes usually has a poor performance in terms of precision [11, 10]. So, many recent research works focus on how to generate short compact hash codes [8, 11, 10, 13, 17]

for each sample with only one hash table or few hash tables, which can still keep good precision. A well-known method for hashing with compact codes is spectral hashing (SH)[11], which has been shown to achieve much better performance than LSH in applications with compact hash codes[11].

In this paper, we also aim at developing a hashing method to generate compact hash codes. However, previous hashing methods with compact codes, e.g., [11, 10, 8], only consider and optimize the search accuracy. Search time, which is also an important factor, is usually not discussed explicitly. In this paper, we develop a new hashing algorithm with a joint framework to explicitly optimize both search accuracy and search time together. To the best to our knowledge, our paper presents the first result on such explicit joint optimization.

Moreover, most previous hashing methods assume data is represented in a vector format and can be embedded as points in a vector space. However, in applications involving computer vision, multimedia, biology, or Web, etc., structured data are quite common [27, 25]. Though several recent previous kernel hashing methods, e.g.,[13, 14, 12] have discussed hashing for non-vectors, but they all need a Mercer kernel function. In our proposed method, any similarity/proximity function (including kernel function of course) is applicable, and thus broadens the possible applications of our method.

The main contributions of our work are

1. We analyze and model the search time, and prove that it can be minimized if the hashing buckets are perfectly balanced, i.e., contain equal number of samples. A maximum entropy criterion and equivalently a minimum mutual information criterion are proposed to achieve the bucket balancing.

2. A joint optimization framework, which considers both the search accuracy and the search time, is provided.

3. Relaxation and implementation techniques are proposed such that the joint optimization problem can be solved efficiently, which outperforms the state-of-the-art hashing methods with compact codes.

4. The proposed method is generalized to handle data of general formats including graphs, sequences, sets, etc. besides vectors, with any kernel function or similarity/proximity function defined.

We have evaluated our algorithm on several databases with a data size ranging from several thousand to one million. Our proposed method significantly outperforms several stat-of-the-art approaches for all the data sets/tasks, including implementations with both single hash table and multiple hash tables.

2. Modeling search accuracy and search time

2.1. Background and notations

We first assume our data are vectors. The extension to non-vector data will be discussed in section 3.5. Suppose feature matrix for training data X is a d by N matrix. d is the dimension of feature, and N is the number of training data. Denote X_p as the p-th column of X, which is the feature vector for p-th data in the training set. Like in most machine learning problems, the training samples $X_p, p = 1, ..., N$ are assumed to be i.i.d. sampled from a random feature vector x.

Suppose M is the number of hash bits, and

$$H : R^d \times \{-1, 1\}^M$$

represents the hash function we are looking for, which maps a d-dimension vector to M binary bits. Intuitively, H consists of M functions $H_j : R^d \times \{-1, 1\}, j = 1, ..., M$, each of which maps a d-dimension vector to 1 binary bit.

Denote $Y_p = H(X_p)$, the hash bits for X_p, which is a M-dimensional vector. Since $\{X_p, p = 1, ..., N\}$ are i.i.d. sampled from a random vector x, $\{Y_p = H(X_p), p = 1, ..., N\}$ are i.i.d. samples from $y = H(x)$. Here y is a M-dimension random binary vector. y_k is denoted as k-th dimension of y. There are M bits, so y has 2^M outcomes:

$$a_i \in \{-1, 1\}^M, i = 1, ..., 2^M$$

Each a_i is associated with a unique M-bit hash code and thus represents a distinct hash bucket. There are 2^M buckets in total. Denote N_i as the number of training samples in bucket i, for $i = 1, ..., 2^M$.

2.2. Similarity preserving for search accuracy

For two data samples X_p and X_q in the training set, suppose W_{pq} is the similarity between X_p and X_q. Similarity W_{pq} can come from feature similarity or label consistency, etc., depending on the applications. The only requirement for W is the symmetry.

To obtain a good search accuracy, hashing methods usually have a common property: similarity preserving, i.e., similar points are supposed to have similar hash codes with small hamming distance. One popular approach to preserver similarity is to minimize the following [11]:

$$D(Y) = \sum_{p=1,...,N} \sum_{q=1,...,N} W_{pq} ||Y_p - Y_q||^2 \qquad (1)$$

where Y is the set of all Y_p. With this criterion, samples with high similarity, i.e., larger W_{pq} , are supposed to have similar hash codes, i.e., smaller $||Y_p - Y_q||^2$.

2.3. Minimal mutual information criterion for search time

Generally speaking, the search process for hash based methods would consist of 3 steps:

1. Compute the hash bits for the query.
2. Find the bucket with the same hash bits as the query.
3. Load samples into memory which are in the selected bucket. Rerank them if necessary.

It is easy to see that similarity preserving alone does not guarantee a good hash/serach method. For example, when the similarity matrix W contains only non-negative elements, as an extreme case, one can always assign the same bits for every data, and hence $D(Y)$ would be 0 and hence minimized. However, in this case, for every query, all the data would be returned, which is the worst case of search time, and actually equals linear scan. So the objective of hashing should not focus on similarity preserving only, but aim at a good tradeoff between search accuracy and search time. We will first model the search time in this section, and then provide the joint optimization framework to achieve tradeoff between search accuracy and time in section 3.1.

Note that the main difference of search time among hash methods is determined by the difference of time complexity for step 3, since the time complexity of step 1 and 2 is independent of the database size. Moreover, for large scale applications, usually, samples can not be pre-stored into memory, so step 3 require harddisk operations, while step 1 and 2 only involve memory operations. So the time complexity of step 3 is the main factor determining the search time, which depends on the number of samples in the selected buckets.

The following proposition shows that the time complexity of step 3 is minimized if all buckets contain the same number of samples, which leads to a minimum mutual information criterion.

Proposition 1: Both the worst case and the average case of time complexity in step 3 can be minimized if the buckets are perfectly balanced, i.e., every bucket contains the same number of samples, which happens when $Entropy(y)$ is maximized, or equivalently, mathematical expectation $E(y_k) = 0$ for $k = 1, ..., M$ and the mutual information $I(y_1, ..., y_k, ..., y_M)$ is minimized.

Proof of Proposition 1: Recall that N_i is the number of training samples in bucket i, for $i = 1, ..., 2^M$. The worst case of processing time in step 3 happens when the selected bucket is the largest bucket, i.e., the bucket with largest N_i. So the worst case can be minimized if the buckets are perfectly balanced, i.e., every bucket contains equal number of samples, in other words, $N_i = \frac{N}{2^M}$, for $i = 1, ..., 2^M$.

Further more, since the offline training data and new query are i.i.d. sampled, then the probability of the query point falls into bucket i is approximated by the frequency $p_i \approx \frac{N_i}{N}$, and moreover the processing time for all samples in bucket i is $S_i = cN_i$, where c is a constant that is related to the time of processing (e.g., loading or displaying) one individual sample. So the expected time to search one bucket is: $\sum_{i=1}^{2^M} p_i S_i \approx \frac{c}{N} \sum_{i=1}^{2^M} N_i^2$ which only depends on $\sum_{i=1}^{2^M} N_i^2$. To minimize the average time complexity in step 3 equals to minimizing $\sum_{i=1}^{2^M} N_i^2$, which can be achieved by $N_i = \frac{N}{2^M}$ for $i = 1, ..., 2^M$, considering $\sum_{i=1}^{2^M} N_i = N$. In sum, both the worst case and the average case of time complexity in step 3 can be minimized if $N_i = \frac{N}{2^M}$ for $i = 1, ..., 2^M$.

Note that $Entropy(y) = \{-\sum_{i=1}^{2^M} P(y = a_i) \log P(y = a_i)\} = \{-\sum_{i=1}^{2^M} \frac{N_i}{N} \log(\frac{N_i}{N})\}$. It is easy to see that $N_i = \frac{N}{2^M}$ for $i = 1, ..., 2^M$, if and only if $Entropy(y)$ gets its maximum value.

As shown in [19],

$$Entropy(y) = \sum_{k=1}^{M} Entropy(y_k) - I(y_1, ..., y_k, ..., y_M) \tag{2}$$

where $I()$ is the mutual information.

So $Entropy(y)$ would be maximized, if $\sum_{k=1}^{M} Entropy(y_k)$ is maximized and $I(y_1, ..., y_k, ..., y_M)$ is minimized. Moreover, note that y_k is a binary random variable. If the mathematical expectation $E(y_k) = 0$, half samples would have bit $+1$ and the other half would have bit -1 for y_k, which means $Entropy(y_k) = 1$, and is maximized.

In conclusion, if $E(y_k) = 0, k = 1, ..., M$ and $I(y_1, ..., y_k, ..., y_M)$ is minimized, $Entropy(y)$ would be maximized, and the search time would be minimized. This completes the proof of Proposition 1.

In practice, many hashing methods are applied with a "multi-probe" strategy, i.e., finding multiple buckets within a small hamming distance or sharing the most reliable hash bits of the query, instead of finding only one bucket with the same hash bits. It is easy to see that the analysis of the worst case search time is still valid in this case.

Moreover, note that if $I(y_1, ..., y_k, ..., y_M)$ is minimized, it means $y_1, ..., y_k, ..., y_M$ are independent [1]. So minimiz-

[1] Independence among hash bits are mentioned in spectral hashing[11], but it does not relate independence to search time, and moreover, there is no actual formulation, derivation or algorithm to achieve the independence.

ing mutual information criterion is also to provide the most compact and least redundant hash codes.

3. Formulation and relaxation

3.1. Formulation of hashing with joint optimization of search accuracy and time

Note that $E(y_k) = 0, k = 1, ..., M$ means $E(y) = 0$, which can further be rewritten as $\sum_{p=1}^{N} Y_p = 0$ with samples $\{Y_p, p = 1, ..., N\}$. By incorporating the similarity preserving term $D(Y)$ for search accuracy and the mutual information criterion for search time, the problem of joint optimization for search accuracy and time together can now be formulated as:

$$\begin{aligned} &\min_{H} I(y_1, ..., y_k, ..., y_M) \\ &s.t., \sum_{p=1}^{N} Y_p = 0 \\ &D(Y) \leq \eta \\ &Y_p = H(X_p) \end{aligned} \tag{3}$$

We first parameterize H, so that it can be optimized more easily. For simplicity, we first assume data are in vector format, and H is a linear function with a sign threshold, i.e.,

$$H(x) = sign(T^T x - b) \tag{4}$$

and later on, we will provide a generalized version in section 3.5 to handle data with general format. Here T is a projection matrix of $d \times M$ and b is a vector.

Even with the parameterizations of H, the problem in (3) is difficult to optimize (e.g., non-differential), and hence some relaxation is needed. In the following discussion, we show how to relax equation (3) with the parameterized H.

3.2. Relaxation for $D(Y)$

First of all, recall that $Y_p = H(X_p) = sign(T^T X_p - b)$. A traditional relaxation as in many algorithms (e.g., [11]), is to remove the binary constraint by ignoring the $sign()$ function such that $D(Y)$ is differentiable. In other words, $D(Y)$ is relaxed as:

$$\sum_{p,q=1,...,N} W_{pq} ||(T^T X_p - b) - (T^T X_q - b)||^2 = \sum_{k=1}^{M} T_k^T C T_k \tag{5}$$

where $C = XLX^T$. Here L is the Laplacian matrix $L = D - W$. D is a diagonal matrix, $D_{p,p} = \sum_{q=1}^{N} W_{p,q}$.

Moreover, $\sum_{p=1}^{N} Y_p = 0 \Rightarrow \sum_{p=1}^{N} (T^T X_p - b) = 0$, so $b = \frac{1}{N} T^T \sum_{p=1}^{N} X_p$.

3.3. Relaxation for minimizing $I(y_1, ..., y_k, ..., y_M)$

Denote $z_k = T_k^T x$, where T_k is the k-th column of T. So $y_k = sign(z_k - b)$.

It is easy to see that if z_k are independent, y_k would be independent too. Hence if $I(z_1, ..., z_k, ..., z_M) = I(T_1^T x, ..., T_k^T x, ..., T_M^T x)$ is minimized, $I(y_1, ..., y_k, ..., y_M)$ would be minimized. So we will minimize $I(z_1, ..., z_k, ..., z_M) = I(T_1^T x, ..., T_k^T x, ..., T_M^T x)$ instead of $I(y_1, ..., y_k, ..., y_M)$ in equation (3).

In the field of ICA, independence or mutual information are well studied for a long time. As discussed in [19], minimizing $I(z_1, ..., z_k, ..., z_M)$ can be well approximated as minimizing $C_0 - \sum_{k=1}^{M} ||g_0 - E(G(z_k))||^2$, which equals maximizing

$$\sum_{k=1}^{M} ||g_0 - \frac{1}{N} \sum_{p=1}^{N} G(T_k^T X_p)||^2 \tag{6}$$

under the constraint of whiten condition, i.e.,

$$\begin{aligned} &E\{z_k z_j\} = \delta_{kj} \\ &\Rightarrow E\{(T_k^T x)(T_j^T x)\} = T_k^T E\{xx^T\} T_j = T_k^T \Sigma T_j = \delta_{kj} \end{aligned} \tag{7}$$

for $1 \leq k, j \leq M$.

Here C_0 is a constant, $E()$ means the expectation, $G()$ is some non-quadratic functions such as $G(u) = -e^{-u^2/2}$, or $G(u) = \log\cosh(u)$, etc., and g_0 is a constant. $\delta_{kj} = 1$, if $k = j$; $\delta_{kj} = 0$, if $k \neq j$. $\Sigma = E(xx^T)$.

3.4. Similarity Preserving Independent Component Analysis (SPICA)

In sum, after relaxation with equation (5), (6), and (7), the problem in equation (3) can now be formulated as:

$$\begin{aligned} &\max_{T_k, k=1...M} \sum_{k=1}^{M} ||g_0 - \tfrac{1}{N} \sum_{p=1}^{N} (G(T_k^T X_p))||^2 \\ &s.t., T_k^T \Sigma T_j = \delta_{kj}, 1 \leq k, j \leq M \\ &\sum_{k=1}^{M} T_k^T C T_k \leq \eta \end{aligned} \tag{8}$$

where $C = XLX^T$ and $\Sigma = E(xx^T)$. The hash bits for X_p can be computed as $Y_p = sign(T^T X_p - b)$, for $p = 1, ..., N$, where $b = \frac{1}{N} T^T \sum_{p=1}^{N} X_p$.

Surprisingly, after relaxation steps mentioned above the solution becomes quite intuitive. We call this method SPICA (Similarity Preserving Independent Component Analysis), because it incorporates a similarity preserving term into the Fast-ICA formulation[19].

3.5. Generalization–GSPICA

In practice, many applications involve structured data in the forms of graphs, trees, sequences, sets, or other formats. For such general data types, usually certain kernel functions are defined to compute the data similarities, e.g., [27, 25]. Moreover, even if the data are stored in the vector format, many machine learning solutions benefit from the use of domain-specific kernels, for which the underlying data embedding to the high-dimensional space is not known explicitly, namely only the pair-wise kernel function is computable. We can obtain the kernelized version of SPICA to deal with data of general format by parameterizing the hash functions as:

$$y = H(x) = sign(T^T K_x - b) \tag{9}$$

where $\mathrm{K}_x = [K(x, Z_1), ..., K(x, Z_i), ..., K(x, Z_P)]^T$. Here K is the kernel function, and $Z_i, i = 1, ..., P$ are some landmark samples, which for example can be a subset chosen from the original N samples. Usually $P \ll N$.

Denote $z_k = T_k^T K_x$. With similar relaxation and derivation, we can get

$$\begin{aligned} &\max_{T_k, k=1...M} \sum_{k=1}^{M} ||g_0 - \tfrac{1}{N} \sum_{p=1}^{N} (G(T_k^T K_{X_p}))||^2 \\ &s.t., \sum_{k=1}^{M} T_k^T C T_k \leq \eta \\ &T_k^T \Sigma T_j = \delta_{kj}, 1 \leq k, j \leq M \end{aligned} \tag{10}$$

where

$$\mathrm{K}_{X_p} = [K(X_p, Z_1), ..., K(X_p, Z_P)]^T$$

Moreover $b = \frac{1}{N} T^T \sum_{p=1}^{N} K_{X_p}$ and $Y_p = sign(T^T K_{X_p} - b)$, for $p = 1, ..., N$. Here, $C = K_{P\times N}(D - W)K_{P\times N}^T$. $K_{P\times N}$ is defined as

$$(K_{P\times N})_{p,i} = K(Z_p, X_i),\ p = 1, \cdots, P,\ i = 1, \cdots, N. \tag{11}$$

And $\Sigma = E\{K_x {K_x}^T\} = \frac{1}{N} \sum_{p=1}^{N} K_{X_p} K_{X_p}^T = \frac{1}{N} K_{P\times N} K_{P\times N}^T$.

In equation(10), one can see that Mercer condition is unnecessary for function K. Actually, any similarity function is applicable. We call the method in equation (10) Generalized SPICA (GSPICA), which can handle both vector data and structured data with any kernel function or similarity/proximity function K defined.

4. Optimization

4.1. Optimization algorithm

The optimization problem in both equation (8) and (10) is nonconvex. It is not trivial to obtain a fast algorithm to solve them efficiently, especially when the data set is very large. Inspired by the work in [19, 20], here we provide a fast and efficient approximate method to solve the problems of equation (8) and (10). The workflow of the optimization is described in algorithm 1 with details. The method is shown to converge quite fast and performs well in the extensive experiments to be described later. The details of derivation for the algorithm can be found in the supplement material. Note that γ in algorithm 1 is equivalent to parameter η in (8) and (10). Actually, $\gamma = 0$ means $\eta = \infty$. Larger γ is equivalent to smaller η.

4.2. Complexity and Scalability

In algorithm 1 for SPICA and GSPICA, the bottleneck of time complexity is the computation of XWX^T or $K_{P\times N} W K_{P\times N}^\top$ in step 3. When we have a large scale data set, what may consist of millions of samples, XWX^T or $K_{P\times N} W K_{P\times N}^\top$ would be very expensive to compute, with a time complexity of $O(dN^2)$ and $O(PN^2)$ respectively.

One way to overcome the computation complexity is to use a sparse W. Generally speaking, one can always sample a small subset of training samples to compute similarity matrix, so that W is sparse, and hence the time complexity of XWX^T or $K_{P\times N} W K_{P\times N}^\top$ is acceptable.

Another approach is by low rank representation/approximation for W such that $W = RQR^\top$, where R and Q are low rank matrix. In this case, $K_{P\times N} W K_{P\times N}^\top$ can be computed as: $K_{P\times N} W K_{P\times N}^\top = (K_{P\times N} R) Q (K_{P\times N} R)^\top$ which involves small matrices only. There are several ways to obtain the low rank approximation, for example, we can choose W as $W = XX^\top$ if X are normalized data, or apply Nyström algorithm [23] to get a low rank approximation for W. Moreover, when the W is defined by some "shift-invariant" kernel functions such as $W(i, j) = e^{-||X_i - X_j||^2/\sigma^2}$, we can apply the kernel linearization technique [24] to approximate W as $W = ZZ^\top$, where Z are the random Fourier Features as in [24]. Note that we don't need to compute or store W, but only compute or store the low rank matrix.

With the speed up, algorithm 1 would take about $O(d^2N)$ or $O(P^2N)$ for SPICA or GSPICA respectively, which is close to state-of-the-art methods like spectral hashing[11] ($O(d^2N)$) or OKH [17] ($O(P^2N)$).

5. Experiments

5.1. Experiment setup

We compare our GSPICA algorithm with several state-of-the-art methods, including spectral hashing (SH) [11], locality sensitive hashing(LSH) and kernelized locality sensitive hashing (KLSH)[12].

All algorithms are compared using the same number of hash bits. For a fair comparison, we always use the

Algorithm 1 Workflow for optimization of SPICA.

Input: data $X_p, p = 1, ..., N$, similarity matrix W, the number of required bits: M .
(By replacing X_p with K_{X_p} and X with $K_{P\times N}$, we can obtain the optimization for GSPICA.)
Output: hash functions to generate M bits for each sample, i.e., $Y_p = H(X_p) = sign(T^T X_p - b)$
Workflow:

1. Compute $\Sigma = \frac{1}{N}\sum_{p=1}^{N} X_p {X_p}^T$; Apply SVD to Σ, $\Sigma = \Omega\Lambda\Omega^T$;

2. $Q = \Omega_M \Lambda_M^{-\frac{1}{2}}$, where Λ_M is a diagonal matrix consisting of M largest eigen values of Λ, Ω_M is the corresponding column of Ω

3. Compute $C = XLX^T = X(D - W)X^T$, Compute $\widetilde{C}$=$Q^T C Q$

4.

for $k = 1, ..., M$ **do**
  if $k = 1$ **then**
    $B = I$
  else
    apply QR decomposition to matrix $[\widetilde{T}_1, ...\widetilde{T}_{k-1}]$ to get matrix B, such that $[\widetilde{T}_1, ...\widetilde{T}_{k-1}, B]$ is a full-rank orthogonal matrix.
  end if
  $A = B^T \widetilde{C} B, \widehat{X}_p = B^T \widetilde{X}_p = B^T Q^T X_p$
  Random intialize w
  repeat
    $\beta = \frac{1}{N}\sum_{p=1}^{N} (w^T \widehat{X}_p G'(w^T \widehat{X}_p)) - \gamma w^T A w.$
    $w^+ = w - [\frac{1}{N}\sum_{p=1}^{N} (G''(w^T \widehat{X}_p))I - \beta I - \gamma A]^{-1} [\frac{1}{N}\sum_{p=1}^{N} (\widehat{X}_p G'(w^T \widehat{X}_p)) - \beta w - \gamma A w].$
    $w = w^+ / ||w^+||.$
  until converge
  $\widetilde{T}_k = Bw$
end for

5. $T_k = Q\widetilde{T}_k$ and $b = \frac{1}{N} T^T \sum_{p=1}^{N} X_p$

6. For any sample X_p, compute its M hash bits $Y_p = sign(T^T X_p - b)$

same kernel function with the same parameters (if any), when kernels are used in methods including GSPICA and KLSH. The same number of landmark samples are used for GSPICA and KLSH. And moreover, the number of landmark points are set close to the number of feature dimensions, so that GSPICA and SH would have almost the same indexing time. The parameter γ is chosen with a validation set, which is independent of the training set or the query set.

5.2. Evaluation metrics

For evaluation, we will first compare the precision-recall curve, which is one of the most papular evaluation methods in large scale retrieval research.

We also report comparison of accuracy-time tradeoff for different hashing methods. Search accuracy is represented by recall rate, i.e., percentage of groundtruth neighbors found. However, direct comparison of machine time for each algorithm is not practical, since different implementation (e.g., different programming languages) may result in varied search times of the same method. So in our experiments, search time is represented via the number of retrieved samples in the selected buckets. By this, we try to provide an unbiased comparison of search time.

5.3. Experiment results

1 million web image data set

This is a data set consisting of $1M$ web images downloaded from flickr web site: www.flickr.com. 512 dimension gist features[22] are extracted for each image. RBF kernel is used for this data set. 32 hash bits are used for all the hashing methods.

To get the precision or recall, we need to obtain groundtruth of the true nearest neighbors for each query sample. Similar to the previous works [11, 12], we establish such groundtruth by choosing the top samples (e.g.,top 100 samples) found via linear scan.

In Figure 1, we first show the comparison of precision-recall curves. GSPICA performs significantly better than other methods, confirming its superiority on search performance. Then we report accuracy-time comparison for different hashing methods. GSPICA also achieves a much better tradeoff between search accuracy and time. For instance, at the same search time (1 k retrieval samples), the recall of our method is several times higher than other methods.

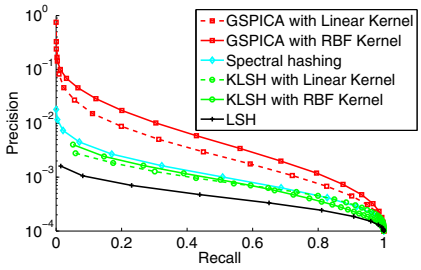
In Figure 2, some example query images and the top 5 search results are also provided. The results of the proposed method are confirmed to be much better than others.

100K Photo Tourism image patch data set with multi hash tables

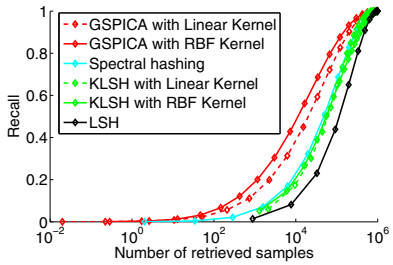
One key property of LSH and its variations like KLSH is the capacity to create multiple hash tables to improve the recall. Though only GSPICA with single hash table is discussed above, it can be easily extended to use multiple hash tables, for example, by using a subset of training set for each table.

We test our GSPICA method with multi hash tables on Photo Tourism image patch set[26].

In our experiment, we use 100K patches, which are extracted from a collection of Notre Dame pictures. $1K$ patches are randomly chosen as queries, $90K$ are used as training set to learn the hashing function. For each patch, 512 dimension gist features [22] are extracted. The task is

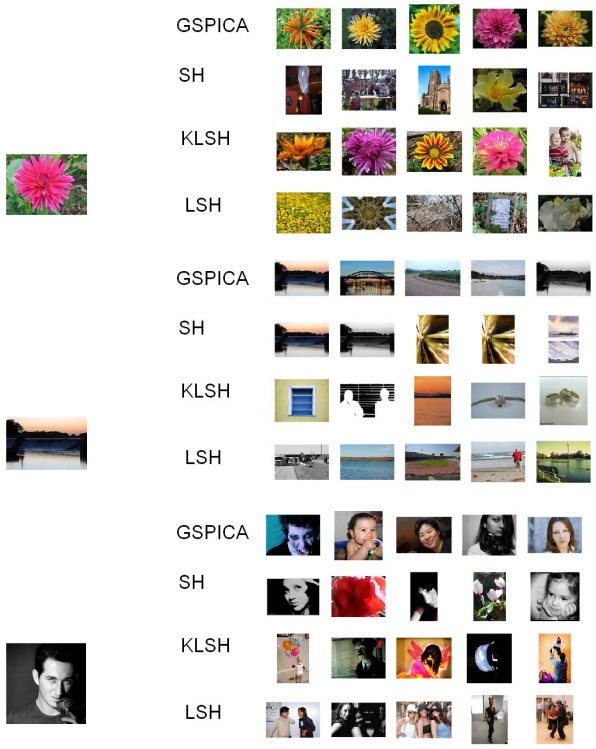


(a) Precision-recall curve on 1M web image data



(b) Accuracy-time comparison on 1M web image data

Figure 1. Search results on 1M web image data set with 32 hash bits. In (a), the comparison of precision-recall curve is provided. In (b), comparison of accuracy-time curve is shown where recall represents search accuracy, and the number of retrieved samples in selected buckets represents search time. Graphs are best viewed in color.



(a) Queries (b) Top 5 search results.

Figure 2. On 1M web image data set, example query images and top 5 search results of GSPICA, SH, KLSH, and LSH ranked by hamming distance with 32 hash bits are shown. Note that we are using gist features, so color information is not considered.

to identify the neighbors, i.e., near-duplicate patches, in the training set for each query patch. For each patch, its near-duplicates are used as the groundtruth. In our experiments, we randomly sample 10,000 training samples to compute 48 bits for each hash table. The same procedure are also done to create multi tables for spectral hashing.

The results are shown in Figure 3. By using multi hash tables, the recall of our GSPICA method is improved. Moreover, GSPICA works significantly better than LSH, KLSH or spectral hashing, no matter with a single hash table or multi hash tables.

We also explores how the change of experiment setting, e.g., the number of landmark samples P, or the parameter γ in algorithm 1, would affect our results. As shown in Figure 4, the proposed method is quite insensitive and stable to reasonable change of P and γ. And not surprisingly, the performance increases slowly with P.

6. Conclusion

In this paper, we develop a new hashing algorithm with joint optimization of search accuracy and search time together, and support data of general formats Our methods significantly outperforms several state-of-the-art hashing algorithms over diverse test set and data types.

Our proposed technique, SPICA and GSPICA, try to find independent (and hence least redundant) projections while keeping the data similarity at the same time, which is highly related to problems like dimension reduction, subspace learning, etc. We will explore application of SPICA for such tasks and other related domains in our future work.

Acknowledgments

"This work has been supported in part by NSF awards #CNS-07-51078 and #CNS-07-16203."

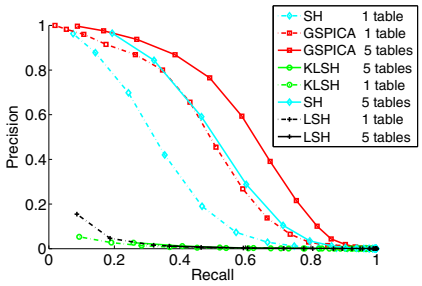
References

[1] J. Freidman, J. Bentley, and A. Finkel. An Algorithm for Finding BestMatches in Logarithmic Expected Time. *ACM Transactions on Mathematical Software*, page 209-226, 1977.

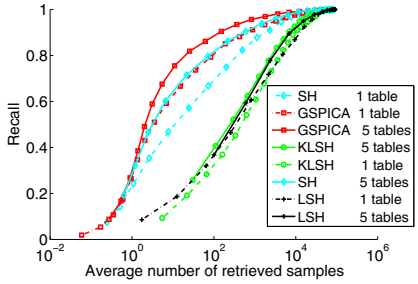
[2] J. Uhlmann. Satisfying general proximity / similarity queries with metric trees. *Information Processing Letters*, page 175-179, 1991.

[3] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. *In Proceedings of STOC*, 1998.

[4] M. Charikar. Similarity estimation techniques from rounding algorithms. *In Proceedings of STOC*, 2002.

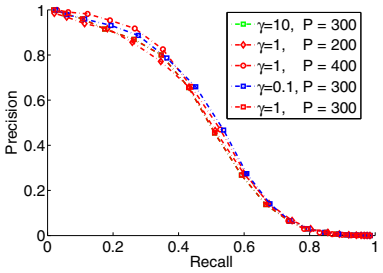


(a) Precision-recall curve on Photo Tourism data



(b) Accuracy-time comparison on Photo Tourism data

Figure 3. Search results on 100K Photo Tourism image patch data with 48 hash bits for each hash table. In (a), the comparison of precision-recall curve is provided. In (b), comparison of accuracy-time curve is shown where recall represents search accuracy, and the number of samples in selected buckets represents search time. Graphs are best viewed in color.



(a) Precision-recall curve for different experiment setting

Figure 4. Search results on 100K Photo Tourism image patch data with 48 hash bits for one hash table with different number of landmark samples P and parameter γ in algorithm 1. Note that the green curve with square marks are covered by other curves and can not be seen. As shown, the proposed algorithm is quite stable and not sensitive to reasonable change of P and parameter γ. Graphs are best viewed in color.

[5] M. Datar, N. Immorlica, P. Indyk, and V. Mirrokni. Locality sensitive hashing scheme based on p-Stable distributions. *In Proceedings of Symposium on Computational Geometry (SOCG)*, 2004.

[6] K. Grauman and T. Darrell. Pyramid match hashing: sub-Linear time indexing over partial correspondences. *In Proceedings of CVPR*, 2007.

[7] P. Jain, B. Kulis, and K. Grauman. Fast image search for learned metrics. *In Proceedings of CVPR*, 2008.

[8] R. Salakhutdinov and G. Hinton. Semantic hashing. *In Proceedings of SIGIR*, 2007.

[9] R. Salakhutdinov and G. Hinton. Learning a nonlinear embedding by preserving class neighbourhood structure. *In Proceedings of AI and Statistics*, 2007.

[10] A. Torralba, R. Fergus, and Y.Weiss. Small Codes and Large Image Databases for Recognition. *In Proceedings of CVPR*, 2008.

[11] Y. Weiss, A. Torralba, and R. Fergus. Spectral hashing. *In Proceedings of NIPS*, 2008.

[12] Brian Kulis and Kristen Grauman. Kernelized Locality-Sensitive Hashing for Scalable Image Search. *In Proceedings of ICCV*, 2009.

[13] Brian Kulis and Trevor Darrell. Learning to hash with binary reconstructive embeddings. *In Proceedings of NIPS*, 2009.

[14] M. Raginsky and S. Lazebnik. Locality sensitive binary codes from shift-invariant kernels. *In Proceedings of NIPS*, 2009.

[15] Ruei-Sung Lin David A. Ross Jay Yagnik. SPEC Hashing: Similarity Preserving algorithm for Entropy-based Coding *In Proceedings of CVPR*, 2010.

[16] Yadong Mu, Jialie Shen, Shuicheng Yan. Weakly-Supervised Hashing in Kernel Space *In Proceedings of CVPR*, 2010.

[17] Junfeng He, Wei Liu and Shih-Fu Chang. Scalable Similarity Search with Optimized Kernel Hashing *In Proceedings of KDD*, 2010.

[18] Shumeet Baluja and Michele Covell. Learning to hash: forgiving hash functions and applications. *Data Mining and Knowledge Discovery*, 2008.

[19] Aapo Hyvarinen and Erkki Oja. Independent Component Analysis: Algorithms and Applications. *Neural Networks*, page 411-430, 2000.

[20] Alper T. Erdogan. On the Convergence of ICA Algorithms With Symmetric Orthogonalization. *IEEE Transactions on Signal Processing*, 2009.

[21] Xiaofei He, and Partha Niyogi. Locality Preserving Projections. *In Proceedings of NIPS*, 2003.

[22] A. Oliva and A. Torralba. Modeling the shape of the scene: a holistic representation of the spatial envelope. *International Journal on Computer Vision*, 2001.

[23] Christopher Williams , Matthias Seeger. Using the nystrom method to speed up kernel machines. *In Proceedings of NIPS*, 2001.

[24] Ali Rahimi and Benjamin Recht. Random Features for Large-Scale Kernel Machines. *In Proceedings of NIPS*, 2007.

[25] S. Lazebnik, C. Schmid and J. Ponce. Beyond bags of features, spatial pyramid matching for recognizing natural scene categories. *In Proceedings of CVPR*, 2006.

[26] S. Winder and M. Brown. Learning Local Image Descriptors. *In Proceedings of Computer Vision and Pattern Recognition*, 2007.

[27] Nino Shervashidze, Karsten M. Borgwardt. Fast subtree kernels on graphs. *In Proceedings of NIPS*, 2009.