Supervised Hashing with Leared Metric

Yisong Li
ShanghaiTech University, SIST
319 Yueyang Road, Shanghai 200031
liys@shanghaitech.edu.cn

Hai Cheng
ShanghaiTech University, SIST
319 Yueyang Road, Shanghai 200031
chenghai@shanghaitech.edu.cn

Abstract

Locality-sensitive hashing is a efficient algorithm to find the approximate nearest neighbour. Data-dependent hashing schemes which utilize supervised information have better performance than classical random hashing scheme. In this paper, we manage to learn a kernel function in terms of Mahalanobis distance metric and combine the kernelized-based supervised hashing scheme to obtain a better performance than the state-of-art supervised hashing scheme. The simulation results show that our scheme is superior.

1. Introduction

The Internet has brought us a wealth of data, large amounts of data are available. But even with the rapid growth of computer performance, we don't have the processing power to search this amount of data by brute force. Thus it is important to design algorithms for nearest-neighbor search. which remain efficient even as the number of points or the dimensionality of the data grows large. The nearest-neighbor (NN) problem occurs in the literature under many names, including the best match or the post office problem.The problem is of significant importance to several areas of computer science, including pattern recognition, searching in multimedia data, vector compression, computational statistics, and data mining.

Firstly, we have to define how to measure the "similarity" between two points, and there are lots of ways to define the "similarity", such as Euclidean distance, Hamming distance, Cosine distance. The exact nearest-neighbor search problem in a Euclidean space is defined as follows:

Given a set P of points in a d-dimensional space, construct a data structure which given any query point q finds the point in P with the smallest distance to q. Implement the technology of face recognition in engineering with computer programming and summarize practicial experience.

A naive algorithm for this problem is as follows: given a query q, compute the distance from q to each point in P, and report the point with the minimum distance. This linear scan approach has query time of $O(dn)$. This is tolerable for small data sets, but is too inefficient for large ones. Our purpose is to design an algorithm for this problem that can efficiently reduct the computational complexity. Many efficient solutions have been discovered for the case when the points lie in a space of constant dimension. For example, if the points lie in the plane, the nearest-neighbor problem can be solved with O(log n) time per query, using only $O(n)$ storage. Unfortunately, as the dimension grows, the algorithms become less and less efficient. More specifically, their space or time requirements grow exponentially in the dimension. In particular, the nearest-neighbor problem has a solution with $O(d^{O(1)}logn)$ query time, but using roughly $n^{O(d)}$ space.

The lack of success in removing the exponential dependence on the dimension led many researchers to conjecture that no efficient solutions exist for this problem when the dimension is sufficiently large. At the same time, it raised the question: Is it possible to remove the exponential dependence on d, if we allow the answers to be approximate? The approximate nearest neighbor search problem is defined as follows.

Given a set P of points in a d-dimensional space, construct a data structure which given any query point q, reports any point within distance at most c times the distance from q to p, where p is the point in P closest to q.

There are many approximate nearest-neighbor algorithms that are more efficient than the exact ones, even though their query time and/or space usage is still exponential in the dimension.There are three common data structures to deal with ANN: kd-trees , balltrees, and locality-sensitive hashing (LSH)[1].

2. Related Work

In thise section we review several work in efficient search algorithms and their application to image retrieval.

Data structures using spatial partitions and recursive hyperplane decomposition(e.g. k-d trees) provide an efficient method to search low-dimensional data.By building a tree of objects, we can start at the top node when given a query,

ask if our query object is to the left or to the right of the current node, and then recursively descend the tree. If the tree is properly constructed, this solves the query problem in O(log N) time, where N is the number of objects. In a one- dimensional space, this is a binary search. In a multi-dimensional space, this idea becomes the k-d tree algorithm. The problem with multidimensional algorithms such as k-d trees is that they break down when the dimensionality of the search space is greater than a few dimensionswe end up testing nearly all the nodes in the data set and the computational complexity grows to O(N).

However, those classical method don't apply to high-dimensional data. Recently, randomized approximate near search algorithm have been proposed to preserve accuracy while query complexity is under control. Locality-sensitive hashing offers sub-linear time search by hashing highly similar examples together in a hash table. LSH functions that accommodate Hamming distance, inner products, l_p norms and learned Mahalanobis metrics[2] have all been developed in prior work.Because of theoretical guarantees that original metrics are asymptotically preserved in the Hamming (code) space with increasing code length, LSH-related methods usually require long codes to achieve good precision. Nonetheless, long codes result in low recall since the collision probability that two codes fall into the same hash bucket decreases exponentially as the code length increases.

In contrast to the random (data-independent) hash scheme(LSH and so on), many data-dependent hashing have been proposed recently. Through encoding high-dimensional data points into compact codes, nearest neighbor search can be accom- plished with a constant time complexity as long as the neighborhood of a point is well preserved in the code space. In addition, compact codes are particularly useful for saving storage in gigantic databases.

A kind of data-dependent hashing scheme is supervised hashing, which utilize supervised information to attain higher search accuracy. *Kernel-Based Supervised Hashing*(KSH) was proposed in 2012 by Liu et al[6]. It take advantage of supervised information to revise hash function based on KLSH.

As for as we can survey, no one has manage to relate metric learning with supervised hashing to obtain a better performance. Therefore, we try to combine kernel-based supervised hashing with kernel(metric) learning to construct a new hashing scheme.

3. Several Hashing Method

3.1. Locality-sensitive hashing

The LSH algorithm is probably the one that has received most attention in the practical context. Its main idea is to hash the data points using several hash functions so as to ensure that, for each function, the probability of collision is much higher for points which are close to each other than for those which are far apart. Then, one can determine near neighbors by hashing the query point and retrieving elements stored in buckets containing that point. It drastically reduce the computational time, at the cost of a small probability of failing to find the absolute closest match.

A family H is called (r, cr, P_1, P_2)-sensitive if for any $p, q \in R^d$

$$\text{If} \quad ||p-q||_2 \leq R \quad then \quad P_H[h_{(q)} = h_{(p)}] \geq P_1 \quad (1)$$

$$\text{If} \quad ||p-q||_2 \geq cR \quad then \quad P_H[h_{(q)} = h_{(p)}] \leq P_2 \quad (2)$$

In order for a LSH family to be useful, it has to satisfy $P_1 > P_2$. Essentially, LSH is an approximation algorithm to relax looking for a polynomial-time algorithm in high dimension nearest problem is NP-HARD. To solve an NP-HARD problem, we have to sacrifice one of the three desired features: optimality, polynomial time, arbitrary instances of the problem. LSH algorithm belongs to the first choice. More general we define a feasible hash function:
There is a s monotonically mapping relationship between any element in set S, such that:

$$sim(a, b) \geq s, \quad then \quad Pr(h(a) = h(b)) \geq p \quad (3)$$

Take cosine similarity for an example:

$$\vec{v} = \{v_1, v_2, ..., v_n\}, \quad sim(\vec{u}, \vec{v}) = \frac{\vec{u} * \vec{v}}{|u||v|} \quad (4)$$

Choose a feasible hash function:

$$h(\vec{v}) = \begin{cases} 1, \ \vec{v} * \vec{x} > 0 \\ 0, \ otherwise \end{cases} \quad (5)$$

Then the recall rate can be writen as:

$$P(s) = 1 - \frac{arccos(s)}{\pi} \quad (6)$$

Repeat several times independently, then we get a $b - bit$ hash value:

$$H(\vec{v}) = (h_1(\vec{v}), \ h_2(\vec{v}), \ ..., \ h_b(\vec{v})) \quad (7)$$

The probability that two element share the same hash value is:

$$P(s) = (1 - \frac{arccos(s)}{\pi})^b \quad (8)$$

Ideally, if all the data is distributed evenly, the search efficiency would be 2^b times than brute force method, but according to equation (8), the recall rate would be smaller. To moderate this trade off, generic data-independent hashing method solve this problem by creating t hash table, e.g. for each element, choose t hash function: $H_i(*)$, the recall rate could be writen as:

$$P(s) = 1 - (1 - (1 - \frac{arccos(s)}{\pi})^b)^t \quad (9)$$

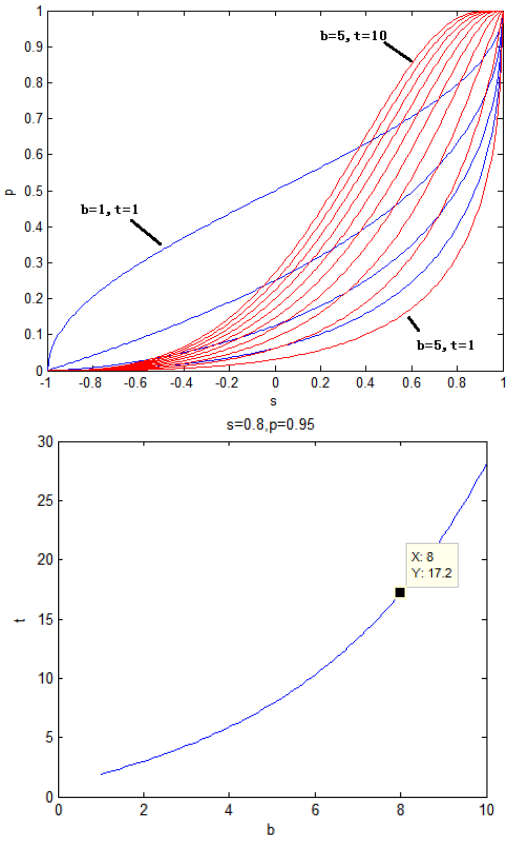

Figure 1. The above figure shows the recall rate-similarity relationship;the below figure show that with specific similarity and recall rate, the relationship between hash bit and hash table

As is shown in the Figure 1, with the parameter t, the steep part of the curve is pre-located. What's more, the recall rate is not so sensitive to similarity on high similarity area,lead to better performance. To achieve high recall rate with smaller hash bit, date-dependent hash methods are investigated.

3.2. Kernelized LSH

Kernelized Locality-Sensitive Hashing(KLSH) is a LSH-based technique for performing fash similarity searches over arbitrary kernel functions.

3.2.1 Kernel Function

Kernel function is defined as

$$\kappa(x, y) = \phi(\boldsymbol{x})^{\mathrm{T}}\phi(\boldsymbol{y}), \tag{10}$$

where $\phi(\cdot)$ is mapping function(embedding function) that maps the low dimension data $\boldsymbol{x}$ to high dimension. By mapping function, we can linearly separate $\phi(\boldsymbol{x})$ in high dimension kernel space even if $\boldsymbol{x}$ can not be linearly separated in low dimension space. Figure 2 illustrate how the kernel work. For many kernel, the mapping function $\phi(\cdot)$ is uncomputable. We can only access the data in kernel space thought kernel function, i.e. the inner product in high dimension kernel space.

3.2.2 Hashing with Kernel

Given a kernel function $\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$ and n objects in database, how to formulate a hashing function on the base of LSH is the key point to apply kernel function in LSH. In LSH, We construct a Gaussian vector $\boldsymbol{r}$ and compute the inner product. The main problem in KLSH is $\phi(\boldsymbol{x})$ may be uncomputable. We need to compute inner product $\boldsymbol{r}^{\mathrm{T}}\phi(\boldsymbol{x})$ via kernel function.

A kind of approach is to construct $\boldsymbol{r}$ as a weighted sum of some elements in database and then utilizes kernel function to compute the inner product in kernel space. This approach was first proposed in [4]. Consider each data point $\phi(\boldsymbol{x}_i)$ as a vector from some underlying distribution $\mathcal{D}$ with mean μ and covariance Σ, which are generally unknown. We define a combination

$$\boldsymbol{z}_t = \frac{1}{t}\sum_{i\in\mathcal{S}}\phi(\boldsymbol{x}_i), \tag{11}$$

where $\mathcal{S}$ is a set of t objects chosen i.i.d from $\mathcal{D}$. From the central limit theorem, for sufficiently large t, the random vector

$$\tilde{\boldsymbol{z}}_t = \sqrt{t}(\boldsymbol{z}_t - \mu) \sim \mathcal{N}(0, \Sigma). \tag{12}$$

Applying a whitening transform, $\Sigma^{-1/2}\tilde{\boldsymbol{z}}_t$ is distributed according to $\mathcal{N}(0, I)$, satisfying the requirement of LSH. Then, we denote the normal Gaussian vector as $\boldsymbol{r} = \Sigma^{-1/2}\tilde{\boldsymbol{z}}_t$, and the desired hash function is given by

$$h(\phi(\boldsymbol{x})) = \begin{cases} 1, & \text{if}\phi(\boldsymbol{x})^T\Sigma^{-1/2}\tilde{\boldsymbol{z}}_t \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{13}$$

As for μ and Σ, we need to estimate them from some finite samples. Choosing a p database objects $\phi(\boldsymbol{x}_1), \cdots, \phi(\boldsymbol{x}_p)$, the mean μ can be estimated as

$$\mu = \frac{1}{p}\sum_{i=1}^{p}\phi(\boldsymbol{x}_i). \tag{14}$$

Similarly, we can estimate covariance matrix Σ over the p samples. Kernel function can be utilized to estimate them implicitly. The detailed discussion can be read at [4]. Here we just give the result

$$h(\phi(\boldsymbol{x})) = \operatorname{sign}\left(\sum_{i=1}^{p}\boldsymbol{w}(i)\left(\phi(\boldsymbol{x}_i)^{\mathrm{T}}\phi(\boldsymbol{x}) - \frac{1}{p}\sum_{j=1}^{p}\phi(\boldsymbol{x}_j)^{\mathrm{T}}\phi(\boldsymbol{x})\right)\right), \tag{15}$$

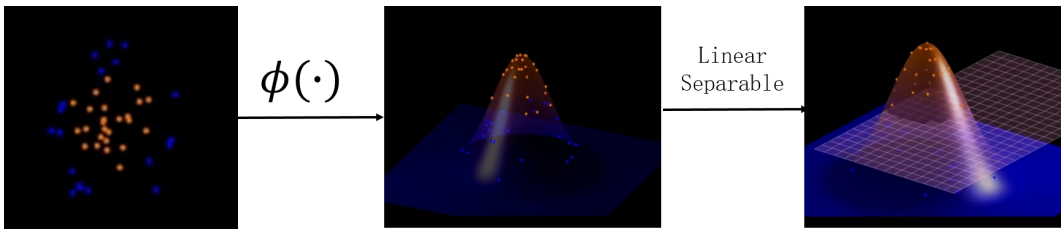

Figure 2. Illustration of Kernel. Mapping function map the blue dots and yellow dots from two dimension space to three dimension space. Then a linear plane can separate yellow dots and blue dots.

where $\boldsymbol{w} = K^{-1/2}\boldsymbol{e}_{\mathcal{S}}$ and $\mathcal{S}$ is a index subset where t points are selected from the p sampled points. Besides, we assume that $\phi(\boldsymbol{x}_i)$ are zero centered implicitly.

3.3. Supervised Hashing with Kernels

Given a data set $\{x_1, ..., x_n\} \in R^d$, we follow the Kernelized Locality-Sensitive Hashing (KLSH) algorithm, choose a hash function:

$$h(x) = sgn(f(x)) \tag{16}$$

where $f(x) : R^d \longrightarrow R$ with kernel k plugged in as follows:

$$f(x) = \sum_{j=1}^{m} k(x_j, x)a_j - b \tag{17}$$

Where x_j is sample uniformly selected at random, a_j is the coefficient we want to optimize by taking leverage on supervised information , k is kernel function tp tackle data that is linear inseparable, b is the bias to take as much information of the original data as possible e.g. to meet $\sum_{i=1}^{n} h(x_i) = 0$, it has the form:

$$b = \sum_{i=1}^{n}\sum_{j=1}^{m} k(x_j, x_i)a_j/n \tag{18}$$

Rewrite the prediction function f as:

$$\begin{aligned} f(x) &= \sum_{j=1}^{m}(k(x_j, x) - \frac{1}{n}\sum_{i=1}^{n} k(x_j, x_i))a_j \\ &= \mathbf{a}^T\bar{\mathbf{k}}(x) \end{aligned} \tag{19}$$

So the hash function is:

$$h(x) = sgn(\mathbf{a}^T\bar{\mathbf{k}}(x)) \tag{20}$$

The supervised information is given in terms of pairwise labels: 1 labels specify similar pairs collected in set M and -1 labels designate dissimilar pairs collected in set C. For semantic data set, the pairs which are in the same class are labled 1, -1 otherwise, so our input is a adjoint matrix S:

$$S(i,j) = \begin{cases} 1, & (x_i, x_j) \in M \\ -1, & (x_i, x_j) \in C \end{cases} \tag{21}$$

Our purpose is generate discriminative hash codes such that similar pairs can be perfectly distinguished from dissimilar pairs by using Hamming distances in the code space. The hamming distance has the formula:

$$D_h(x_i, x_j) = |\{k|h_k(x_i) \neq h_k(x_j), 1 \leq k \leq r\}| \tag{22}$$

Directly optimizing the Hamming distances is non-trivial because of its complex mathematical formula, we utilize the one-to-one correspondence between a hamming distance and a code inner product:

$$\begin{aligned} &code_r(x_i) \circ code_r(x_j) \\ &= |\{k|h_k(x_i) = h_k(x_j), 1 \leq k \leq r\}| \\ &\quad - |\{k|h_k(x_i) \neq h_k(x_j), 1 \leq k \leq r\}| \\ &= r - 2 * |\{k|h_k(x_i) \neq h_k(x_j), 1 \leq k \leq r\}| \\ &= r - 2 * D_h(x_i, x_j) \end{aligned} \tag{23}$$

Then we get the objective function:

$$\min_{H_l \in \{1,-1\}^{l \times r}} = ||\frac{1}{r}H_l H_l^T - S||_F^2 \tag{24}$$

Where:

$$\begin{aligned} H_l &= \begin{bmatrix} h_1(x_1) & \cdots & h_r(x_1) \\ & \cdots & \\ h_1(x_l) & \cdots & h_r(x_l) \end{bmatrix} \\ &= sgn(\bar{\mathbf{k}}_l\mathbf{A}) \end{aligned} \tag{25}$$

Then all we have to do is to optimize (18), cause it's neither convex nor smooth, we use the algorithm form [6], do spectral relaxation [7] as a warm start, do sigmoid smoothing and use Nesterovs gradient method to do optimization.

4. Metric Learning

One of the basic requirements of many machine learning algorithms is to compare two objects by computing a similarity or distance between them. In many cases, standard distance or similarity functions such as the Euclidean distance or cosine similarity are used; for example, in image retrieval applications, the cosine similarity is a standard function to compare two text image features. However, such

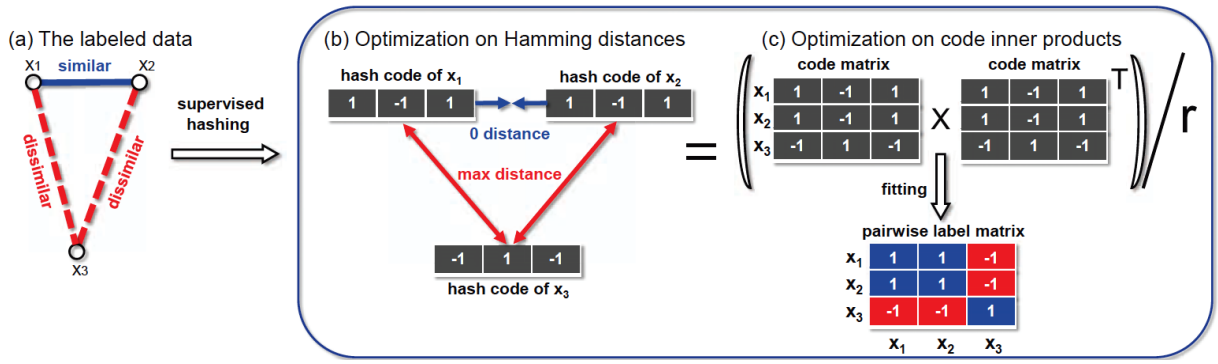

Figure 3. Captured from [6], the one-to-one correspondence between a Hamming distance and a code inner product

standard distance or similarity functions are not appropriate for all problems. To obtain more accurate distance or similarity between two object, we need to learn a more specific distance function, according to the dataset in hand. One approach can be learning a distance metric from the given dataset information, such as pairwise similarity or dissimilarity constraints.

4.1. Mahalanobis distance

Mahalanobis distance is a kind of distance metrics that has shown excellent properties. The Mahalanobis distance between two points $\boldsymbol{x}$, parameterized by a positive definite matrix $\boldsymbol{W}$ is defined as

$$d_{\boldsymbol{W}}(\boldsymbol{x}_i, \boldsymbol{x}_j) = (\boldsymbol{x}_i - \boldsymbol{x}_j)^{\mathrm{T}} \boldsymbol{W} (\boldsymbol{x}_i - \boldsymbol{x}_j). \tag{26}$$

Factorizing the parameterized matrix $\boldsymbol{W} = \boldsymbol{G}^{\mathrm{T}}\boldsymbol{G}$, the the squared distance can be represented as

$$d_{\boldsymbol{W}}(\boldsymbol{x}, \boldsymbol{y}) = \|\boldsymbol{G}\boldsymbol{x} - \boldsymbol{G}\boldsymbol{y}\|_2^2. \tag{27}$$

The distance can be viewed as learning a linear transformation of the data and then measuring the squared euclidean distance in the transformed space.

To learn the resulting metric, we assume that we are given constraints on the desired distance function, i.e. pairwise similarity and dissimilarity constraints are given over the datathat is, pairs of points that should be similar under the learned metric, and pairs of points that should be dissimilar under the learned metric. Such constraints are natural in many settings; for example, given class labels over the data, points in the same class should be similar to one another and dissimilar to points in different classes.

4.2. Mahalanobis Distance in Kernel Space

As discussed in the previous section, we often map our original dataset in low dimension to the kernel space in high dimension to get linearly separable dataset. The Mahalanobis distance in kernel space is defined as

$$d_{\boldsymbol{W}} = (\phi(\boldsymbol{x}) - \phi(\boldsymbol{y}))^{\mathrm{T}} \boldsymbol{W} (\phi(\boldsymbol{x}) - \phi(\boldsymbol{y}). \tag{28}$$

The problem we faced in kernel space is we cannot access the data explicitly. What we know about the data is only through the kernel function:

$$\kappa_0(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x})^{\mathrm{T}} \phi(\boldsymbol{y}). \tag{29}$$

Therefore, the metric learning problem is equivalent to a kernel learning problem. We need to learn a new kernel function

$$\kappa(\boldsymbol{x}, \boldsymbol{y}) = \phi(\boldsymbol{x})^{\mathrm{T}} \boldsymbol{W} \phi(\boldsymbol{y}) \tag{30}$$

when given input kernel function $\kappa_0(\boldsymbol{x}, \boldsymbol{y})$ and prior data information. It is useful to linearly separate dataset by the prior information when combining kernel trick and metric learning.

4.3. Logdet Metric Learning

The LogDet divergence between two positive definite matrix $\boldsymbol{W}, \boldsymbol{W}_0 \in \mathbb{R}^{d \times d}$ is defined as

$$D_{ld}(\boldsymbol{W}, \boldsymbol{W}_0) = \mathrm{tr}(WW_0^{-1}) - \mathrm{logdet}(WW_0^{-1}) - d. \tag{31}$$

What we want to find is $\boldsymbol{W}$ that is closest to $\boldsymbol{W}_0$ measured $D_{ld}(\boldsymbol{W}, \boldsymbol{W}_0)$ when satisfies constraints. When given similarity constraints set $\mathcal{S}$ and dissimilarity constraints $\mathcal{D}$, the metric learning problem is formulated as

$$\begin{aligned} \min_{\boldsymbol{W} \succeq 0} \quad & D_{ld}(\boldsymbol{W}, \boldsymbol{W_0}) \\ \text{s.t.} \quad & d_{\boldsymbol{W}}(\boldsymbol{x}_i, \boldsymbol{x}_j) \le u, \quad (i,j) \in \mathcal{S}. \\ & d_{\boldsymbol{W}}(\boldsymbol{x}_i, \boldsymbol{x}_j) \le u, \quad (i,j) \in \mathcal{D}. \end{aligned} \tag{32}$$

The above optimization problem was firstly considered in [2].LogDet has many important properties that make it useful for machine learning and optimization, including scale-invariance and preservation of the range space.

4.4. Logdet Metric Learning in Kernel Space

We define a kernel matrix $\boldsymbol{K}$ first.

$$\boldsymbol{K}(i,j)) = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^{\mathrm{T}} \boldsymbol{W} \phi(\boldsymbol{x}_j) \tag{33}$$

Then the Mahalanobis distance in kernel space can be represented as

$$d_{\boldsymbol{W}}(\phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) = \boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_i) + \boldsymbol{K}(\boldsymbol{x}_j, \boldsymbol{x}_j) - 2\boldsymbol{K}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \mathrm{tr}(\boldsymbol{K}(\boldsymbol{e}_i - \boldsymbol{e}_j)(\boldsymbol{e}_i - \boldsymbol{e}_j)^{\mathrm{T}}), \tag{34}$$

where $\boldsymbol{e}_i$ is the i-th canonical basis vector. The Metric/Kernel learning problem can be represented as

$$\begin{aligned} \min_{\boldsymbol{K} \succeq 0} \quad & D_{ld}(\boldsymbol{K}, \boldsymbol{K}_0) \\ \text{s.t.} \quad & \mathrm{tr}(\boldsymbol{K}(\boldsymbol{e}_i - \boldsymbol{e}_j)(\boldsymbol{e}_i - \boldsymbol{e}_j)^{\mathrm{T}}) \leq u \quad (i,j) \in \mathcal{S}, \\ & \mathrm{tr}(\boldsymbol{K}(\boldsymbol{e}_i - \boldsymbol{e}_j)(\boldsymbol{e}_i - \boldsymbol{e}_j)^{\mathrm{T}}) \geq l \quad (i,j) \in \mathcal{D}. \end{aligned} \tag{35}$$

The kernel learning problem(35) was first proposed in [5]. Let $\boldsymbol{X} = [\phi(\boldsymbol{x}_1), \phi(\boldsymbol{x}_2), \cdots, \phi(\boldsymbol{x}_n)]$ be a d by n matrix. Then kernel matrix $\boldsymbol{K}_0 = \boldsymbol{X}^{\mathrm{T}}\boldsymbol{X}$. We have the following theorem.

Theorem 1. *Let $\boldsymbol{W}^*$ be the optimal solution to problem* (32) *and let $\boldsymbol{K}^*$ be the optimal solution to problem* (35)*. The two optimal solutions have the following relation:*

$$\begin{aligned} \boldsymbol{K}^* &= \boldsymbol{X}^{\mathrm{T}}\boldsymbol{W}^*\boldsymbol{X}, \\ \boldsymbol{W}^* &= \boldsymbol{I} + \boldsymbol{X}\boldsymbol{M}^*\boldsymbol{X}^{\mathrm{T}}, \end{aligned} \tag{36}$$

where $\boldsymbol{M} = \boldsymbol{K}_0^{-1}(\boldsymbol{K}^* - \boldsymbol{K}_0)\boldsymbol{K}_0^{-1}$.

The theorem is proved in [3].

4.5. Kernel Learning Algorithm

Since problem (35) can be infeasible, we need to incorporate slack variables into formulation to prevent infeasibility. The slack formulation is:

$$\begin{aligned} \min_{\boldsymbol{K}, \xi} \quad & D_{ld}(\boldsymbol{K}, \boldsymbol{K}_0) + \gamma D_{ld}(textdiag(\xi), \mathrm{diag}(\xi_0) \\ \text{s.t.} \quad & \mathrm{tr}(\boldsymbol{K}(\boldsymbol{e}_i - \boldsymbol{e}_j)(\boldsymbol{e}_i - \boldsymbol{e}_j)^{\mathrm{T}}) \leq \xi_{ij} \quad (i,j) \in \mathcal{S}, \\ & \mathrm{tr}(\boldsymbol{K}(\boldsymbol{e}_i - \boldsymbol{e}_j)(\boldsymbol{e}_i - \boldsymbol{e}_j)^{\mathrm{T}}) \geq \xi_{ij} \quad (i,j) \in \mathcal{D}. \end{aligned} \tag{37}$$

The parameter γ controls the tradeoff between minimizing LogDet divergence and satisfying the constraints. The elements of ξ_0 are set to be u for corresponding similarity constraints and l for corresponding dissimilarity constraints. The optimal approach to the slack formulation (37) was discussed detailedly in [3]. Here we just present the learning algorithm. With different slack variable γ, we can get different kernel matrix $\boldsymbol{K}$. We need to tune γ carefully according to the data on hand.

Algorithm 1 Kernel Learning with LogDet Divergence

Require: $\boldsymbol{K}_0$: input n $\times$ n kernel matrix, $\mathcal{S}$: set of similar pairs, $\mathcal{D}$: set of dissimilar pairs, u, l : distance threshold, γ : slack parameter

Ensure: $\boldsymbol{K}$: output kernel matrix

$\boldsymbol{K} \leftarrow \boldsymbol{K}_0, \lambda_{ij} \leftarrow 0, \forall i, j$
$\xi_{ij} \leftarrow u$ for $(i,j) \in \mathcal{S}$; $\xi_{ij} \leftarrow l$ for $(i,j) \in \mathcal{D}$
while Not convergence **do**
  Select one constraint $(i,j) \in \mathcal{S}$ or $\mathcal{D}$
  $p \leftarrow \boldsymbol{K}(\boldsymbol{e}_i - \boldsymbol{e}_j)(\boldsymbol{e}_i - \boldsymbol{e}_j)^{\mathrm{T}}$
  $\delta \leftarrow$ if $(i,j) \in \mathcal{S}$,-1 otherwise
  $\alpha \leftarrow \min(\lambda_{ij}, \frac{\delta\gamma}{\gamma+1}(\frac{1}{p} - \frac{1}{\xi_{ij}}))$
  $\beta \leftarrow \delta\alpha/(1 - \delta\alpha p)$
  $\xi_{ij} \leftarrow \gamma\xi_{ij}/(\gamma + \delta\alpha\xi_{ij})$
  $\lambda_{ij} \leftarrow \lambda_{ij} - \alpha$
  $\boldsymbol{K} \leftarrow \boldsymbol{K} + \beta\boldsymbol{K}(\boldsymbol{e}_i - \boldsymbol{e}_j)(\boldsymbol{e}_i - \boldsymbol{e}_j)^{\mathrm{T}}$
end while

5. Experiments

We do experiment on CIFAR-10 dataset. The CIFAR-10 dataset consists of 60000 32×32 colour images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images. We uniformly randomly sample 200 images from each class respectively, constituting 2K labeled subsets for training:

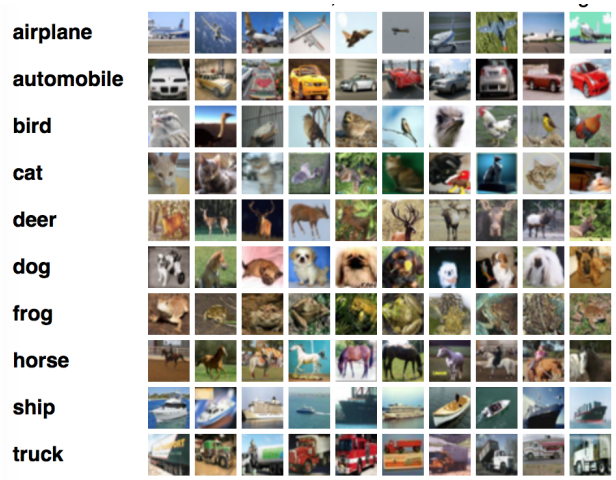

Figure 4. CIFAR-10 dataset

First we set training data number to 1000, and compare the 10-neighbor precision with 8 to 48 range hash bit.

We also test the how the precision changes with increase of training data number.

We run our code on a 2.7 GHz Intel Core i5 laptop, and record the cost time.

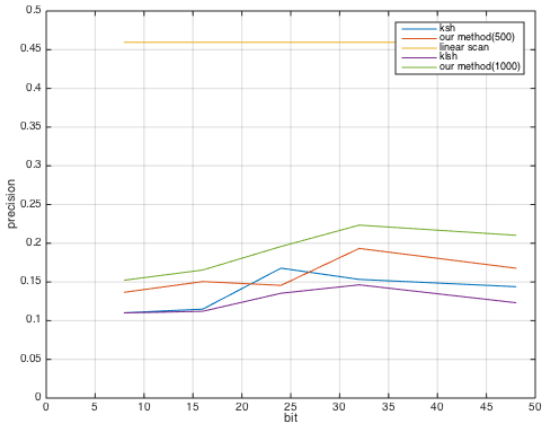

Figure 5. Mean precision of top-10 hamming distance hash lookup

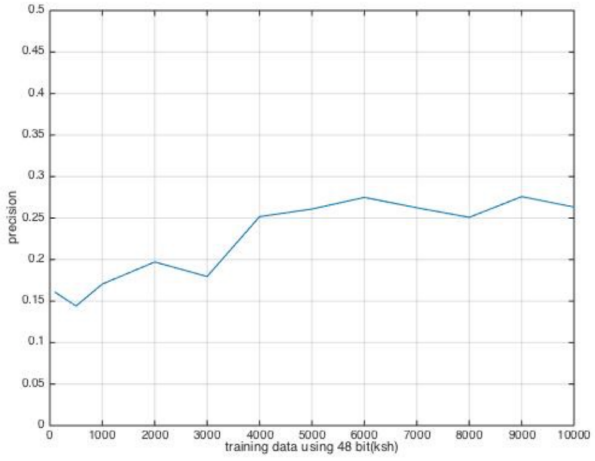

Figure 6. Mean precision with the number of training data

Method	Query time	Training time
linear scan	0.12	-
klsh	$4.55 * 10^{-5}$	-
ksh	$4.55 * 10^{-5}$	27.69
our method	$4.55 * 10^{-5}$	$27.69 + 698$

Table 1. Training and query time recorded in second

6. Conclusion and Discussion

Based on KSH and KlSH method, we come up with a novel idea to improve the precision performance on CIFAR-10 dataset, besides optimizing the parameter matrix, we also optimize the kernel matrix. Theoretically, it will work definitely, in our experiment, our method surpasses any other method's performance, but the precision level is so poor, $5 - 10$ percentage evenly smaller than state-of-art, we accuse it to our wrong code, but unfortunately, we have not fix it yet.

References

[1] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

[2] J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon. Information-theoretic metric learning. In *Proceedings of the 24th international conference on Machine learning*, pages 209–216. ACM, 2007.

[3] P. Jain, B. Kulis, J. V. Davis, and I. S. Dhillon. Metric and kernel learning using a linear transformation. *The Journal of Machine Learning Research*, 13(1):519–547, 2012.

[4] B. Kulis and K. Grauman. Kernelized locality-sensitive hashing for scalable image search. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2130–2137. IEEE, 2009.

[5] B. Kulis, M. Sustik, and I. Dhillon. Learning low-rank kernel matrices. In *Proceedings of the 23rd international conference on Machine learning*, pages 505–512. ACM, 2006.

[6] W. Liu, J. Wang, R. Ji, Y.-G. Jiang, and S.-F. Chang. Supervised hashing with kernels. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2074–2081. IEEE, 2012.

[7] W. Liu, J. Wang, S. Kumar, and S.-F. Chang. Hashing with graphs. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 1–8.