NOKMeans: Non-Orthogonal K-means Hashing

Xiping Fu(✉), Brendan McCane, Steven Mills, and Michael Albert

Department of Computer Science, University of Otago, Dunedin, New Zealand
{xiping,mccane,steven,malbert}@cs.otago.ac.nz

Abstract. Finding nearest neighbor points in a large scale high dimensional data set is of wide interest in computer vision. One popular and efficient approach is to encode each data point as a binary code in Hamming space using separating hyperplanes. One condition which is often implicitly assumed is that the separating hyperplanes should be mutually orthogonal. With the aim of increasing the representation capability of the hyperplanes when used for indexing, we relax the orthogonality assumption without forsaking the alternate view of using cluster centers to represent the indexing partitions. This is achieved by viewing the data points in a space determined by their distances to the hyperplanes. We show that the proposed method is superior to existing state-of-the-art techniques on several large computer vision datasets.

1 Introduction

Finding nearest neighbor points is of wide interest in several areas of computer vision including feature matching [1], image retrieval [2] and object recognition [3]. For example, in content-based image retrieval (CBIR), the task is to retrieve similar images when given a query image. This is often done by representing all of the images as points in a specific space, and then retrieving the nearest neighbor points as similar images. Naive exhaustive searching is linear in the number of images in the collection and becomes infeasible for very large collections. Even specialised data structures such as KD-trees deteriorate to linear search complexity or worse if the dimensionality of the data is large [4]. Since computer vision problems often have very large collections and high dimensional data, approximation algorithms are of interest.

A popular approach for approximate nearest neighbor search is to represent each feature as a point (binary code/hash code) in Hamming space which enables fast retrieval and reduces storage space. As a case in point, one SIFT feature takes $128 \times 4 = 512$ bytes if it is stored as a 128 dimensional floating point vector, while it only occupies $128/8 = 16$ bytes if it is represented as a point in 128D Hamming space. Further, calculating the distance between two points in Hamming space is very quick since it only involves a bitwise XOR operation followed by a bit count.

Locality sensitive hashing (LSH) [5,6] pioneered the use of hashing for fast approximate nearest neighbor searching. In 1998, Indyk et al. introduced the concept of a locality sensitive function family. Each function in this family has

D. Cremers et al. (Eds.): ACCV 2014, Part I, LNCS 9003, pp. 162–177, 2015.
DOI: 10.1007/978-3-319-16865-4_11

the property that it can preserve the similarity between data points. Thus, by randomly choosing functions from this family, a binary code for each data point can be constructed. In order to preserve the similarity between data points when they are represented in Hamming space, however, a large number of bits are usually needed [7]. LSH has been generalized to Euclidean-LSH [8] for different similarity measures, and to shift kernel hashing [9] and Circulant Binary Embedding [10] by introducing different locality sensitive function families.

Machine learning techniques have also been used to design data dependent hashing algorithms. Borrowing ideas from manifold learning [11], Spectral Hashing [12] calculates the binary code by embedding the data points into Hamming space. The optimal embedding is determined by the following characteristics: neighbourhood relationships should be preserved, the code should be balanced (-1's and 1's should occur with roughly equal frequency), and bits should be pairwise independent. Spectral Hashing also leads to data dependent hashing algorithms. Various hashing algorithms have been proposed recently, including ones that use boosting based methods [13–16], exploit the spectral property [7,17,18], utilize the order of distance information [19,20], use supervised information [21–25], and so on.

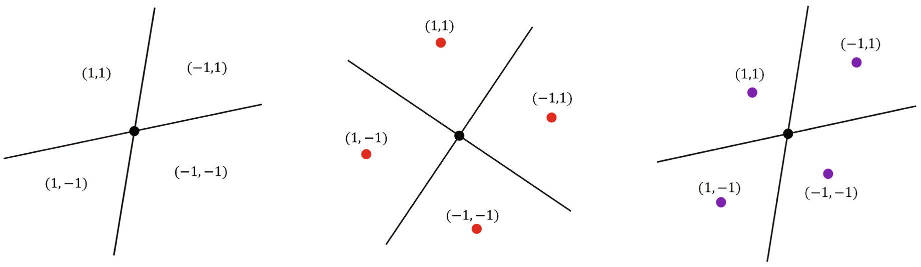

Fig. 1. Visualization of different encoding approaches. The left figure shows two hyperplanes in general position. The space is divided into four partitions. The binary code of a data point in this space can be determined by its relative positions to these hyperplanes. The middle figure shows that when the hyperplanes are mutually orthogonal, we can find a set of indexing centers (red points) whose binary codes have been predefined. The binary code of the points in this space can be determined by assigning the same code to its nearest indexing center. The right figure is a visualization of our proposed method. The purple points are the implicit indexing centers which are viewed in the re-represented space (Color figure online).

Since these algorithms use a binary encoding, the input space is divided into two pieces for each bit. Most of the hashing algorithms address this kind of partition problem by learning a set of hyperplanes which can be viewed either in the original space or in the Reproducing Kernel Hilbert space (RKHS). Each data point is encoded by its relative position to these hyperplanes, -1 for one side of the plane, 1 for the other. The left picture in Fig. 1 shows the space partitioned by two hyperplanes in 2D space. When the hyperplanes are mutually orthogonal, we can find a set of data points (indexing centers) such that the partition of the space (Voronoi diagram) according to this set coincides with the space partitioned by the hyperplanes. The hash code for the indexing centers can be predefined since these points are chosen from the vertices of a hyper-cube [26] or hyper-cuboid [27]. The middle picture in Fig. 1 shows such a set of indexing centers (red points). Thus the encoding process for each data point can also be viewed as assigning the same binary code to its nearest indexing center. Both ITQ [26] and Orthogonal K-means (OKmeans) [27] aim to learn the mutually orthogonal hyperplanes such that the total quantization error between the data point and its indexing center is minimized.

The focus of this paper is to design a compact binary code based on minimizing quantization error. We propose a novel hashing algorithm: Non-Orthogonal K-means (NOKMeans). The essential idea of this algorithm is that increasing the freedom of the separating hyperplanes can lead to a better binary code in the sense of the recall of retrieval performance. We achieve this by relaxing the orthogonality constraints in [26,27] to a near orthogonal assumption. One problem introduced by this relaxation is that the explicit indexing centers cannot be found in the original feature space any more. This is because for any given indexing centers, the space will be divided into Voronoi cells if we index each data point by its nearest indexing center. When the hyperplanes are not mutually orthogonal, it is impossible to find center points such that the hyperplanes are exactly the separating boundaries for the Voronoi diagram. We address this problem by viewing the data points in a re-represented space where we can find specific indexing centers. After encoding each data point into its nearest indexing center, the hyperplanes are exactly the separating boundary of the Voronoi diagram. In the right picture in Fig. 1, the purple points are viewed as a set of indexing points after re-representation.

2 Background

2.1 Notation

Here are the common notations we use throughout the paper. Suppose we want to build a binary code index for the data set $\{x_1, x_2, \cdots, x_N\}, x_i \in \mathcal{R}^D$. Denote $X \in \mathcal{R}^{N \times D}$ as the data matrix where each row is a data point. The binary code for this data set is denoted as $B \in \{-1, 1\}^{N \times d}$ where each row corresponds to a d bit binary code in Hamming space. sign(X) is a function returning a Hamming matrix of equal size to X, with -1 or 1 as entries depending on the sign of the input entries. Assuming the data set is already centered and the hyperplanes pass through the origin, each hyperplane can be fixed by its normal direction. Therefore, the d hyperplanes, which are used to determine the binary code, can be specified by a hyperplane matrix (projection matrix) $A \in \mathcal{R}^{D \times d}$ where each column corresponds to one normal direction. Another view of the space which is partitioned by the hyperplanes is that the space is divided into regions (partitions) where the data points in the same region have the same binary code. We use $\mathbf{1}$ to represent an all ones column vector and I is the identity matrix.

2.2 Related Work

When given a set of indexing centers, the encoding process shares some similarity with the K-mean clustering algorithm. Each data point is encoded into a binary code according to its nearest indexing center which behaves as a cluster center in the K-means algorithm. ITQ [26] aims to find an optimal binary code in the sense of minimal quantization error. Specifically, the data points are preprocessed by centering and then projecting to a low dimensional space by PCA, and then solving an optimization problem. The main idea of the optimization problem is to find a rotation $R \in \mathcal{R}^{d\times d}$ in order to minimize the quantization error between the rotated data points and their corresponding indexing centers. Suppose $V \in R^{N\times d}$ is the preprocessed data, and $B \in \{-1,1\}^{N\times d}$ is the encoding binary matrix. The optimization problem of ITQ is:

$$\begin{aligned} \min \quad & J(R,B) = ||B - VR||_F^2 \\ s.t. \quad & R'R = I \\ & B \in \{-1,1\}^{N\times d} \end{aligned} \tag{1}$$

If we combine the PCA projection matrix P and the final orthogonal matrix R together, we can see that ITQ aims to find a set of mutually orthogonal hyperplanes, and its optimization model is to find minimum quantization error in the subspace obtained from PCA, and its index centers are from the vertices of a rotated d-dimensional hyper-cube in the PCA subspace. The left picture in Fig. 2 shows a visualization of ITQ. Two hyperplanes are used to partition the data points. The red points are the corresponding indexing centers. The data points in the space can be encoded by either its relative position to the hyperplanes or its nearest indexing center.

OKmeans [27] generalizes ITQ by embedding the vertices of a d dimensional hyper-cube in $\mathcal{R}^D$, then the vertices are rotated by $R \in \mathcal{R}^{d\times D}$, scaled by $S \in \mathcal{R}^{d\times d}$ (S is a diagonal matrix) in the corresponding directions and translated by $\mu \in \mathcal{R}^D$. Thus the indexing centers can be viewed as points chosen from the vertices of a rectangular hyper-cuboid. Finally, the optimization objective is modelled as minimizing the quantization error which is formulated as:

$$\begin{aligned} \min \quad & J(R,\mu,B,S) = ||X - \mathbf{1}\mu - BSR||_F^2 \\ s.t. \quad & R'R = I \\ & \mu \in \mathcal{R}^D \\ & B \in \{-1,1\}^{N\times d} \\ & S \in \mathcal{R}^{d\times d}, \qquad S_{i,j} = 0 \text{ if } i \neq j \in \{1,2,\cdots,d\} \end{aligned} \tag{2}$$

If we view $\mu \in \mathcal{R}^D$ as the 'origin' of the data points, the columns of the rotation matrix behave as the normal directions of the corresponding hyperplanes, the regions of the data points which have the same index are separated by these hyperplanes. A visualization of OKmeans is shown in the right picture of Fig. 2. From the visualization of both ITQ and OKmeans, we can see that both of the partitions from ITQ and OKmeans are divided by mutually orthogonal hyperplanes, and the indexing centers of ITQ and OKmeans are chosen from a unit hyper-cube and a rectangular hyper-cuboid respectively.

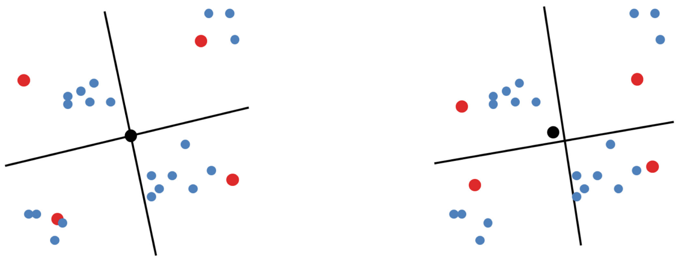
Fig. 2. Visualization of data points encoded by ITQ and OKmeans. The black point is the center of the data points. The red points are the indexing centers. The indexing centers are chosen from the vertices of a square (ITQ) and rectangle (OKMeans) respectively (Color figure online).

3 Proposed Algorithm

As discussed above, the main idea of ITQ and OKmeans is to find a set of mutually orthogonal hyperplanes. The hyperplanes are used to divide the data points into partitions, thus we can index each data point as a point in $\{-1, 1\}^d$ according to its relative position to the hyperplanes. In this work, we investigate the situation when the orthogonality assumption used in ITQ and OKmeans is relaxed. Specifically, we adopt the 'near' mutually orthogonal property which is also used in [24]. One advantage of the relaxation is that it will increase the representation capability of the hyperplanes. The quality of the hash code is closely related to the position of the hyperplanes since the hyperplanes behave as the separating boundary of different index regions. Thus the flexibility of the position of the hyperplanes can lead to smaller overall quantization error. Furthermore, as discussed in [24], the 'near' mutually orthogonal condition is favored since the mutually orthogonal condition has some practical problems even though it is an approximation to the bit independent property. Therefore, to some degree, the near orthogonal constraint is a trade off between independence [12] and representation capability.

When the separating hyperplanes are orthogonal to each other, we can find appropriate indexing centers in the original space. For example, if we view the red points in Fig. 2 as indexing centers, and then index each data point according to its nearest indexing center, we will find that the region of different index partitions is exactly the same as the space partitioned by hyperplanes. When the hyperplanes are not orthogonal, it is impossible to find indexing centers in the original space; when given a set of data points as indexing centers, the resulting segmented space will have a general Voronoi diagram structure and the hyperplanes will not coincide with the boundary diagram when the corresponding hyperplanes are not orthogonal. The left picture of Fig. 3 shows the resulting Voronoi cells for the indexing centers (purple points) in the original space. We can see that it is impossible to find two hyperplanes to separate these cells.

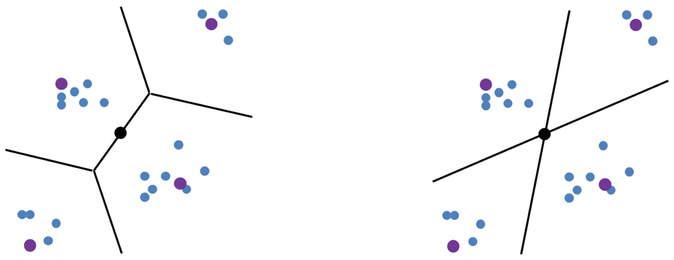

Fig. 3. Visualization of the Voronoi diagrams under different views. The purple points in the left picture are viewed as the indexing centers in the original space. The resulting Voronoi cells can never be separated by two hyperplanes since the indexing centers are in general position rather than from the vertices of a square or a rectangular. The right picture shows the resulting Voronoi cells when the purple points are viewed as the indexing centers in the re-represented space (Color figure online).

In this work, we view the indexing centers in a transformed space used to re-represent the data points. In this way, if we index each data point by its nearest center, the hyperplanes coincide with the boundaries of different index regions. The intuitive idea is as follows: suppose the hyperplanes are fixed, we can re-represent each data point by its relative distance to the hyperplanes. Figure 4 shows one way to re-represent the data points. There are three hyperplanes in the original space. After re-representation, the data point is represented as a point in 3D space. In the new representation space, the vertices $\{-1, 1\}^d$ can be viewed as the indexing centers. If we encode every data point as the binary code of its nearest indexing center, we can see that the hyperplanes are exactly the boundary of the different indexing regions. An example of partitioning with non-orthogonal hyperplanes is shown in Fig. 3, the purple points can be viewed as the indexing centers in the new representation space. One advantage of this view is that the hyperplanes can be viewed as the boundary of the different indexing regions. This re-representation trick allows us to formulate the problem by minimising the quantisation error between data points and indexing centers.

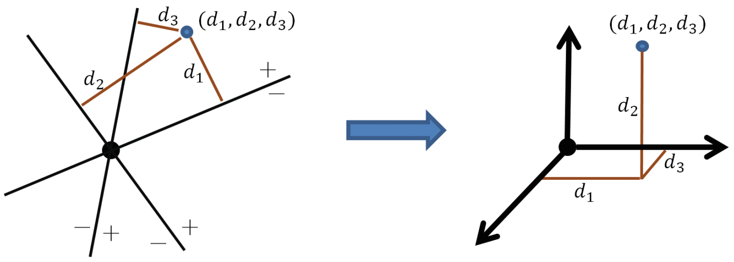

Fig. 4. Visualization of re-representing the data point. The original two dimensional space is partitioned by three hyperplanes, and each data point is represented by its distance to the hyperplanes. We can see that each indexing regions corresponds to one octant (there are 2^3 of them) in the right figure.

3.1 Formulation

When the orthogonal assumption is relaxed to near orthogonal, we propose the following optimization model to learn the hyperplanes which can be used to index the data points:

$$\begin{aligned} \min \quad & J(A,B) = \frac{1}{2N}||XA - B||_F^2 + \frac{\lambda}{4}||A'A - I||_F^2 \\ s.t. \quad & A \in \mathcal{R}^{D\times d} \\ & B \in \{-1,1\}^{N\times d} \end{aligned} \tag{3}$$

In above optimization model, the columns of A behave as the normal directions of the hyperplanes, and $B \in \{-1,1\}^{N\times d}$ is the encoding matrix. When each column of A has a unit norm, the resulting re-represented data points are represented using the distance information of the data point to each hyperplane, i.e. each data point is represented by a vector where each element is corresponding to the signed distance between the data point and one specific hyperplane. When the columns do not satisfy the unit norm property, the new representation can be viewed as a scaling effect on the data points represented by distance information. The first part in the objective function is used to minimize the average quantization error, the second part is used as a regularizer in order to maintain the near orthogonal property, λ is the regularization parameter and the fraction is kept for convenience in further calculation. The constraint on hyperplanes has an effect on the final binary matrix indirectly since it guarantees the balance condition on B to some degree.

For solving the above optimization problem, we use alternating descent to find a locally optimal solution:

Fix. A it is easy to see that $B = \text{sign}(XA)$ is the optimal solution.
Fix. B we use first order gradient descent to update the projection matrix A:

$$\frac{\partial J(A,B)}{\partial A} = \frac{1}{N}X'(XA - B) + \lambda(AA' - I)A \tag{4}$$

Therefore, A can be updated by $A - \gamma \frac{\partial J(A,B)}{\partial A}$. For the step length γ, we use a simple line search strategy, i.e. start with $\gamma = 1$, if the updated A does not improve the cost function, update the step length γ with $s\gamma$. The process is continued until the total cost is reduced.

Algorithm 1. Non-Orthogonal K-means

```
Input: $N$ training data points $x_1, x_2, \cdots, x_N$, $I_{max}$ is the maximum iteration for the
    overall optimization problem and $I_{step}$ is the maximum iteration for finding the
    step length.
Output: The hyperplane matrix $A$
1: Center the training data points, denote the centered data points as $X \subset \mathcal{R}^{D \times N}$,
   and initializing the hyperplane matrix $A$.
2: for $i = 1$ to $I_{max}$ do
3:     Update $B$:
           $B \leftarrow \text{sign}(XA)$
4:     Calculate the gradient direction by (4)
5:     Search the step length $\gamma$ by backtracking line search with at most $I_{step}$ iter-
       ations. If the maximum number of step length search iterations is reached, stop
       training stage.
6:     Update $A$ by:
           $A \leftarrow A - \gamma \frac{\partial J(A,B)}{\partial A}$
7: end for
```

In all of the following experiments, the initialization of A is obtained by the PCA projection times a random rotation matrix which is a common approach used in hashing algorithms, and the s is set to be 0.125. The backtracking line search process takes 10 to 20 iterations to find the appropriate step length. Thus, in order to make the computational complexity of the proposed algorithm measurable, we set the maximum iteration for the line search step as $I_{step} = 50$. When the maximum number of iterations is reached, we stop the training stage since the projection is almost unchanged if we continue to update A by a very small step length. Finally, the maximum iteration I_{max} for the alternative descent method is set to 50 which is the same value used in ITQ and OKmeans.

3.2 Computational Complexity

In each iteration, updating B takes $\Theta(NDd)$. For updating the projection matrix A, it takes two steps. The first step is to calculate the gradient which takes $\Theta(ND^2 + NDd)$. Here $\Theta(ND^2)$ is used for calculating $X'X$. For establishing the step length, in each iteration, we have to check the appropriateness of the current γ, this involves calculating the objective function which takes $\Theta(NDd)$. Suppose the maximum iteration for the overall optimization problem is I and the maximum iteration for finding the step length is I_{step}, thus the overall computational complexity is $\Theta(I_{max}(NDd + ND^2 + NDd + I_{step}(NDd))) = \Theta(I_{max}ND^2 + I_{max}I_{step}NDd)$. $\Theta(ND^2)$ is used for calculating the $X'X$ which can be precalculated, therefore the overall computational complexity of the training stage is $\Theta(ND^2 + I_{max}I_{step}NDd)$ which takes more computation than ITQ and OKmeans ($\Theta(I_{max}(Nd^2 + d^3)$ and ($\Theta(I_{max}(NDd + D^3))$ respectively). During the training stage on our machine (implemented in MATLAB with single core), NOKMeans takes about 2.820 s per iteration when N, D and d are set to 10^5, 128 and 64 respectively, while for the same setting, the ITQ and OKmeans take 0.259 and 0.419 respectively. Since the line search step requires the most computation in the current implementation, the training stage can be sped up by choosing an appropriate initial step length γ and backtracking parameter s or a faster line search algorithm.

Finally, for encoding data and query points, it takes the same computational complexity as most hashing based algorithms, i.e. each data point takes $\Theta(Dd)$ time to compute the binary code. For retrieving the K nearest neighbor points in Hamming distance, we use exhaustive search in all of our experiments since the distance calculation is efficient in Hamming space and it is easy to find the nearest neighbor points due to the distance property which only take values from $\{0, 1, 2, \cdots, d\}$. To speed up this process, one can build a hash table or use fast nearest neighbor searching designed for Hamming space [28].

3.3 Discussion

The proposed method shares some similarity with ITQ. For ITQ, the final projection matrix is the PCA projection matrix P multiplied by the learned rotation matrix R, and, for the proposed algorithm, the projection matrix is learned during the optimization process directly. When the parameter λ is infinite, the projection matrix A in NOKMeans shares the orthogonal property, i.e. $A'A = I$. Nevertheless there are still some differences between these two algorithms even as λ tends to infinity. For example, the quantization error in ITQ is calculated in the PCA subspace, while the quantization error in the proposed method is calculated in the space determined by A which is learned during the optimization process.

Compared to OKmeans, the proposed algorithm also utilizes a rectangular hyper-cuboid to some degree. When the parameter λ is positive, the learned projection matrix can be decomposed into $A = QS$ where each column in $Q \in \mathcal{R}^{D\times d}$ has a unit norm, and S is a diagonal matrix which has a scaling effect in each direction. Notice that the overall quantization error $||XA - B||_F^2 = |S|^2||XQ - BS^{-1}||_F^2$, thus our proposed method can be viewed as re-representing each data point by its distance information to each hyperplane, and then encoding the data point according to the indexing centers from the vertices of a hyper-cuboid. This hyper-cuboid is obtained by scaling the unit hyper-cube by S^{-1} in the corresponding directions. On the other hand, the S in the optimization objective in OKmeans is viewed as an independent variable. One advantage of modeling the scale effect is that the resulting indexing centers will fit better to the data points in the sense of the quantization error, but it also introduces a distortion problem. For example, the Hamming distance between neighboring vertices is always 1, but their Euclidean distance is not fixed due to the scale effect of the hyper-cuboid in different directions. With the regularization parameter λ in our method, the scale value will be constrained to around 1, and therefore the scale distortion is minimal.

4 Experiments

We evaluate the proposed hashing algorithm on four real data sets: SIFT1M [29], SIFT10M, SIFT1B [30], GIST1M [29], and compare the performance of related algorithms: LSH [5], Spectral Hashing [12], ITQ [26], OKmeans [27], on these data sets. SIFT1M, SIFT1B, and GIST1M are three benchmark data sets which are used for testing the performance of different nearest neighbor searching algorithms. In SIFT1M and SIFT1B, each data point is a 128D SIFT feature [31] extracted from Flickr images and INRIA Holidays images [32]. In GIST1M, each data point is a 960D GIST feature which is extracted from the tiny image set of [33], Holidays image set and Flickr1M [34].

SIFT10M is our own dataset. Each data point in this set is a SIFT feature which is extracted from Caltech-256 [35] by the open source VLFeat library [36]. Caltech-256 is a benchmark image data set in computer vision, that features a large number of classes (256) and high intra-class variations in each category. For SIFT10M, the base data points, training data points, and the query points are randomly chosen from all SIFT features extracted from the image set. The true nearest neighbor points are provided by exhaustive nearest neighbor search in 128D Euclidean space. For the other three benchmark data sets, we use the publicly available base points, training points, and query points, and true nearest neighbor information for the query points directly. Detailed information about these data sets including the number of training points, query points and base data points used in our experiments are summarized in Table 1.

Table 1. Datasets which are used for evaluating different approximate nearest search algorithms

Data set	Data type	Dimension	Base points	Training points	Query points
SIFT1M	SIFT feature	128	1,000,000	100,000	10,000
SIFT10M	SIFT feature	128	10,000,000	1,000,000	10,000
SIFT1B	SIFT feature	128	1,000,000,000	1,000,000	10,000
GIST1M	GIST feature	960	1,000,0000	500,000	1,000

4.1 Performance Measurements

We adopt recall as an indicator of retrieval performance since this is an important indicator of retrieval performance and it also has internal relationships with other common measurements. For instance, higher recall often corresponds to higher precision. For each query data point, we retrieve its nearest N data points in the sense of the Hamming distance. Recall@N is the percentage of true nearest neighbor points in the retrieved data set, i.e.:

$$\text{Recall@}N = \frac{\text{\#retrieved true nearest neighbor points}}{\text{\#true nearest neighbor points}}$$

We have varied N from 1 to $N_{max} = 10,000$ to give a better overall picture of performance. In order to evaluate the overall retrieval performance of the hashing algorithm, we introduce the m-Recall (mean recall) measure. m-Recall is calculated as:

$$\text{m-Recall}(\lambda) = \frac{\sum_{i=1}^{N_{max}} Recall_\lambda(i)}{N_{max}}$$

here N_{max} is the maximum retrieved nearest points in Hamming space and $Recall_\lambda(i)$ is the average recall of retrieving i nearest neighbor points for the regularization parameter λ.

m-Recall shares some similarity with mean average precision which is often used to measure the overall performance of retrieval algorithms. The main difference between these two measurements is that N_{max} is set to the total number of data points in mean average precision, while here, N_{max} is relatively small compared to the whole data base. For evaluating the performance of hashing algorithms, the data sets often contain millions or billions of points and retrieving part of the data set is of interest in practice. m-Recall is introduced to evaluate the overall performance when only part of the data set is retrieved.

4.2 Parameter Selection

When using this model to learn the hyperplanes, we have to decide the value of the parameter λ. In order to test the sensitivity of the parameter λ in the optimization, we choose λ from $\{10^1, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7\}$. The m-Recall for different λ is shown in the right image in Fig. 5. From Fig. 5, we can see that, at beginning, the m-Recall increases as λ increases. After reaching the peak, the m-Recall dips moderately as λ keeps increasing. This motivates us to have a strategy to learn the index for different data sets. For each data set, λ is fixed by choosing from $\{10^1, 10^2, 10^3, 10^4, 10^5, 10^6, 10^7\}$ for one specific task, and choosing the parameter such that it reaches peak performance, then use this parameter for the whole data set. In the following experiments, the parameter is fixed as $10^4, 10^5, 10^6, 10^5$ for SIFT1M, SIFT10M, SIFT1B and GIST1M respectively. However it is worth noting that performance is similar for values of λ in the range 10^4–10^6 on all four data sets.

4.3 Results

We have evaluated the proposed algorithm, ITQ, OKmeans, Spectral Hashing and LSH on the four data sets. Figure 6 shows the recall curves of different algorithms for searching nearest neighbor points. For each SIFT feature data set,

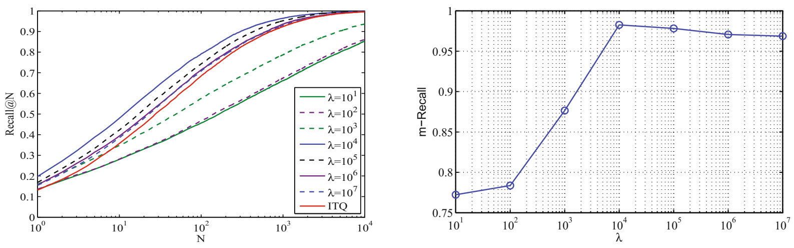

Fig. 5. Retrieval performance on SIFT1M data set with different λ. Each data point is encoded into 128 bits, and the task is to retrieve the nearest neighbor point for each query. The performance of ITQ (red line) is also reported as a baseline (Color figure online).

we report the recall performance when the data points are encoded by 64, 96 and 128 bits respectively. The experimental results enable us to view the performance of different algorithms from two dimensions, i.e. increase the number of bits to encode the SIFT feature and the scale of the data sets. From these two dimensions, we can see that, generally speaking, the more bits used to encode the feature points, the higher the recall. On the other hand, the nearest neighbor point searching problem becomes more difficult for bigger data sets. When each data point is indexed to a 64 bit binary code, the proposed algorithm has a comparable recall performance to the state of art result (OKmeans), and has a much better recall performance than the remaining algorithms. When we use more bits to encode the data set, we find that the proposed algorithm has the highest performance among these algorithms. For the GIST1M data set (Fig. 7), which has the highest number of dimensions among our test scenarios, a similar pattern is observed. From the plot, we can see that the performance gain of our proposed algorithm is larger when the number of encoding bits is increased.

The performance of the proposed algorithm coincides with our model assumption. This is because when few bits are used, the independence property, which leads to the mutually orthogonal condition, plays the main role when designing the hash code. When more bits are used to index the feature points, the mutual orthogonality condition leads to the loss of representation capability. Take the following toy example. Suppose the data points are distributed in a 2D subspace in 3 dimensional space, and we use three bits to encode the data points. If the three hyperplanes are mutually orthogonal, the data points are effectively encoded with two bits since the 2D subspace is partitioned into 4 different index regions. When the orthogonality condition is relaxed, the 2D space can be partitioned into 6 different indexing regions. So when the mutual orthogonality condition is relaxed, the separating capability of the hyperplanes will increase.

As discussed in [18,37], retrieving different numbers of nearest neighbor points affects the performance of searching algorithms. Figure 8 shows the performance of retrieving different K neighbors. Here the K ranges from $\{1, 5, 10, 20, 50, 100\}$, and each data point is encoded as a 256 bit binary code. The top left figure presents the performances of different algorithms for retrieving 1 nearest

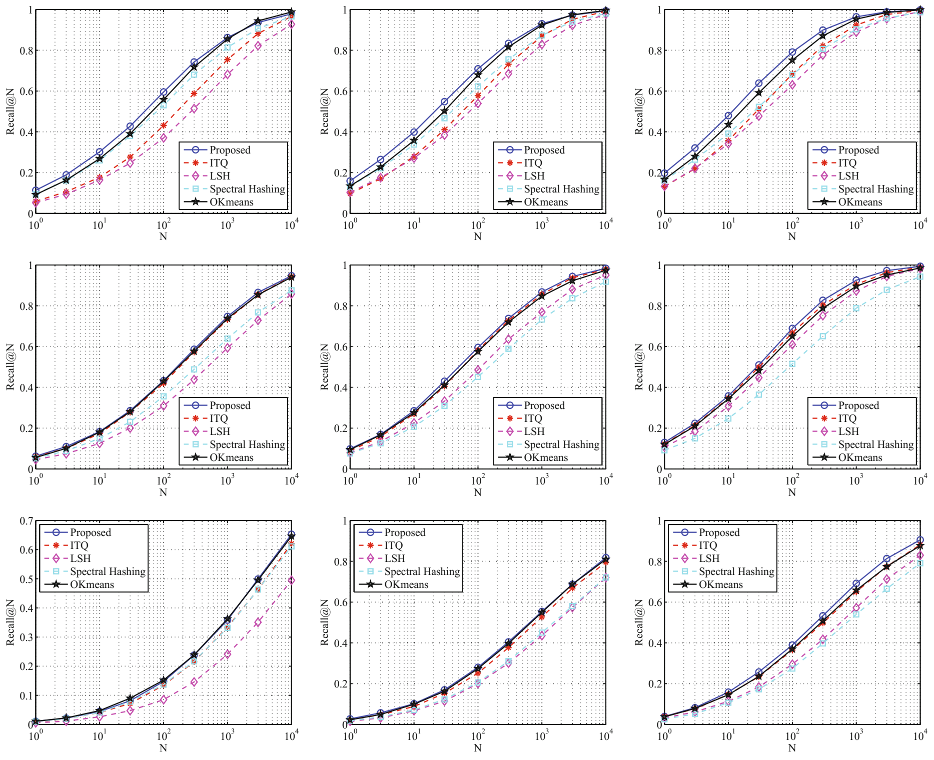

Fig. 6. Retrieval performance on SIFT feature data sets. Different data sets are reported in different rows (SIFT1M, SIFT10M, SIFT1B respectively). For each SIFT feature data set, we show the *Recall@N* when each data point is encoded with 64, 96, 128 bits for columns 1, 2 and 3 respectively.

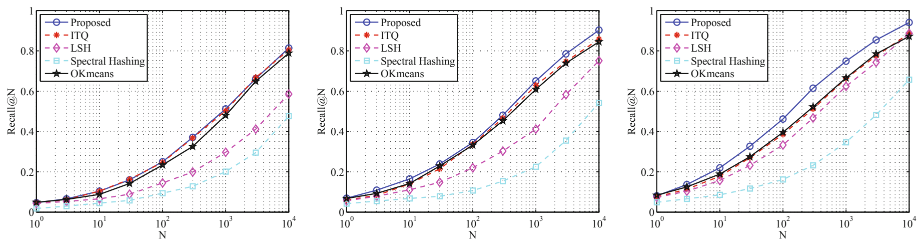

Fig. 7. Retrieval performance on GIST1M data set. The left figure shows the retrieval performance of different algorithms when each data points are encoded into 64 bit binary code. The following two figures show the retrieval performance when using 128 or 256 bits to encode.

neighbor point. The top middle shows the performances of retrieving 5 nearest neighbor points, followed by retrieving 10, 20, 50, and 100 nearest neighbor points. As the trend shows, we can see that the overall recall value keeps going

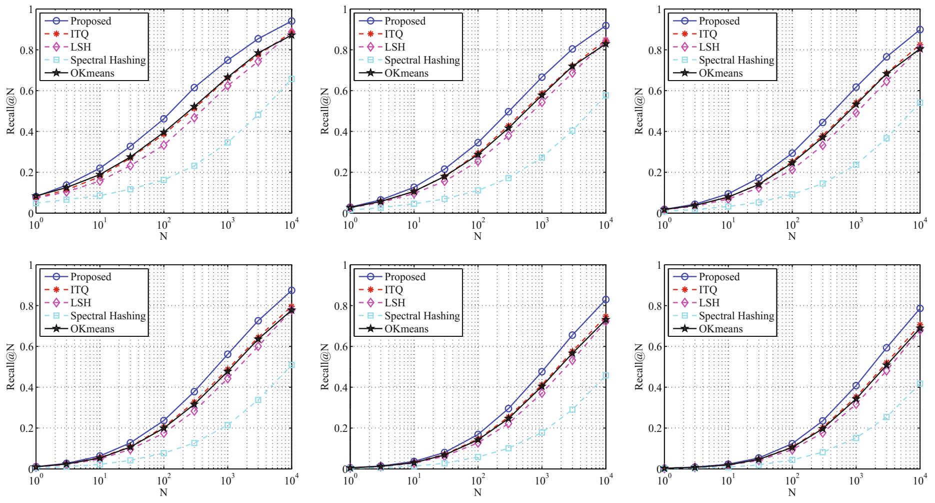

Fig. 8. Retrieval results on GIST1M data set for retrieving different K nearest neighbor points. Here, the K ranges from $\{1, 5, 10, 20, 50, 100\}$, and each GIST feature is encoded by 256 bits.

down. This means that retrieving the K-th nearest neighbor point is more difficult as K increases. Another phenomenon we can see from the plot is that, in order to retrieve more true nearest neighbor points, it is better to retrieve a relatively large number of points in the Hamming distance. For example, if we retrieve 1000 nearest neighbor points in Hamming space, the recall of finding 100 nearest neighbor points is much lower than the recall of searching 1 nearest neighbor point, while for retrieving 10,000 points in Hamming space, the recall gap between different nearest neighbor points is relatively small.

5 Conclusion

In this paper, we have investigated a minimum quantization error based hashing algorithm. Specifically, our focus is on the quantization error of the re-represented data points. In this way, the Voronoi diagram in the original space is the same as the space which is separated by hyperplanes. Compared to the previous quantization based algorithms, the hyperplanes learned in our algorithm are without the constraint that they are mutually orthogonal. We believe this relaxation leads to a better binary code index for large scale high dimensional data. We have tested the proposed algorithm on three benchmark data sets as well as a new SIFT data set. The experimental results shows that our method performs better than current state of the art methods especially when encoding high dimensional data points.

References

1. Brown, M., Lowe, D.: Recognising panoramas. In: ICCV, pp. 1218–1225 (2003)
2. Frome, A., Singer, Y., Sha, F., Malik, J.: Learning globally-consistent local distance functions for shape-based image retrieval and classification. In: ICCV, pp. 1–8 (2007)
3. Torralba, A., Fergus, R., Weiss, Y.: Small codes and large image databases for recognition. In: CVPR pp. 1–8 (2008)
4. Weber, R., Schek, H., Blott, S.: A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In: Proceedings of the 24th VLDB Conference, pp. 194–205 (1998)
5. Indyk, P., Motwani, R.: Approximate nearest neighbors: towards removing the curse of dimensionality. In: Proceedings of the 30th Annual ACM Symposium on Theory of Computing, pp. 604–613 (1998)
6. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: Proceedings of the Thiry-Fourth Annual ACM Symposium on Theory of Computing, pp. 380–388 (2002)
7. Shen, F., Shen, C., Shi, Q., Hengel, A.V.D., Tang, Z.: Inductive hashing on manifolds. In: CVPR, pp. 1562–1569 (2013)
8. Datar, M., Immorlica, N., Indyk, P., Mirrokni, V.S.: Locality-sensitive hashing scheme based on p-stable distributions. In: Symposium on Computational Geometry, pp. 252–262 (2004)
9. Raginsky, M., Lazebnik, S.: Locality-sensitive binary codes from shift-invariant kernels. In: NIPS (2009)
10. Yu, F.X., Sanjiv, K., Gong, Y., Chang, S.F.: Circulant binary embedding. In: ICML (2014)
11. Belkin, M., Niyogi, P.: Laplacian eigenmaps and spectral techniques for embedding and clustering. In: NIPS (2001)
12. Weiss, Y., Antonio, T., Robert, F.: Spectral hashing. In: NIPS, pp. 1753–1760 (2008)
13. Jin, Z.M., Hu, Y., Lin, Y., Zhang, D.B., Lin, S.D., Cai, D., Li, X.: Complementary projection hashing. In: ICCV, pp. 257–264 (2013)
14. Kim, S., Kang, Y., Choi, S.: Sequential spectral learning to hash with multiple representations. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) ECCV 2012, Part V. LNCS, vol. 7576, pp. 538–551. Springer, Heidelberg (2012)
15. Xu, H., Wang, J., Li, Z., Zeng, G., Li, S., Yu, N.: Complementary hashing for approximate nearest neighbor search. In: ICCV, pp. 1631–1638 (2011)
16. Wang, J., Kumar, S., Chang, S.F.: Sequential projection learning for hashing with compact codes. In: ICML, pp. 1127–1134 (2010)
17. Liu, W., Wang, J., Kumar, S., Chang, S.F.: Hashing with graphs. In: ICML, pp. 1–8 (2011)
18. Weiss, Y., Fergus, R., Torralba, A.: Multidimensional spectral hashing. In: Fitzgibbon, A., Lazebnik, S., Perona, P., Sato, Y., Schmid, C. (eds.) ECCV 2012, Part V. LNCS, vol. 7576, pp. 340–353. Springer, Heidelberg (2012)
19. Wang, J., Liu, W., Sun, A., Jiang, Y.: Learning hash codes with listwise supervision. In: ICCV, pp. 3032–3039 (2013)
20. Wang, J., Wang, J., Yu, N., Li, S.: Order preserving hashing for approximate nearest neighbor search. In: Proceedings of the 21st ACM International Conference on Multimedia, pp. 133–142 (2013)

21. Norouzi, M., Fleet, D., Salakhutdinov, R.: Hamming distance metric learning. In: NIPS, pp. 1070–1078 (2012)
22. Norouzi, M., Fleet, D.: Minimal loss hashing for compact binary codes. In: ICML, pp. 353–360 (2011)
23. Kulis, B., Darrell, T.: Learning to hash with binary reconstructive embeddings. In: NIPS, pp. 1042–1050 (2009)
24. Wang, J., Kumar, S., Chang, S.F.: Semi-supervised hashing for scalable image retrieval. In: CVPR, pp. 3424–3431 (2010)
25. Liu, W., Wang, J., Ji, R., Jiang, Y., Chang, S.F.: Supervised hashing with kernels. In: CVPR, pp. 2074–2081 (2012)
26. Gong, Y., Lazebnik, S.: Iterative quantization: a procrustean approach to learning binary codes. In: CVPR, pp. 817–824 (2011)
27. Norouzi, M., Fleet, D.: Cartesian k-means. In: CVPR, pp. 3017–3024 (2013)
28. Norouzi, M., Punjani, A., Fleet, D.: Fast search in hamming space with multi-index hashing. In: CVPR, pp. 3108–3115 (2012)
29. Jegou, H., Douze, M., Schmid, C.: Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell. **33**, 117–128 (2011)
30. Jegou, H., Tavenard, R., Douze, M., Amsaleg, L.: Searching in one billion vectors: re-rank with source coding. In: ICASSP, pp. 861–864 (2011)
31. Lowe, D.G.: Distinctive image features from scale-invariant keypoints. Int. J. Comput. Vision **60**, 91–110 (2004)
32. Jegou, H., Douze, M., Schmid, C.: Improving bag-of-features for large scale image search. Int. J. Comput. Vision **14**, 316–336 (2010)
33. Torralba, A., Fergus, R., Freeman, W.T.: 80 million tiny images: a large database for non-parametric object and scene recognition. IEEE Trans. Pattern Anal. Mach. Intell. **30**, 1958–1970 (2008)
34. Jegou, H., Douze, M., Schmid, C.: Hamming embedding and weak geometric consistency for large scale image search. In: Forsyth, D., Torr, P., Zisserman, A. (eds.) ECCV 2008, Part I. LNCS, vol. 5302, pp. 304–317. Springer, Heidelberg (2008)
35. Griffin, G., Holub, A., Perona, P.: Caltech-256 object category dataset. Technical report, pp. 1–20 (2007)
36. Vedaldi, A., Fulkerson, B.: VLFeat: an open and portable library of computer vision algorithms. In: Proceedings of the International Conference on Multimedia, pp. 1469–1472 (2008)
37. He, K., Wen, F., Sun, J.: K-means hashing: an affinity-preserving quantization method for learning binary compact codes. In: CVPR, pp. 2938–2945 (2013)