Random Subspace for Binary Codes Learning in Large Scale Image Retrieval

Cong Leng, Jian Cheng, Hanqing Lu
National Laboratory of Pattern Recognition
Institute of Automation, Chinese Academy of Sciences
Beijing, China
{cong.leng, jcheng, luhq}@nlpr.ia.ac.cn

ABSTRACT

Due to the fast query speed and low storage cost, hashing based approximate nearest neighbor search methods have attracted much attention recently. Many state of the art methods are based on eigenvalue decomposition. In these approaches, the information caught in different dimensions is unbalanced and generally most of the information is contained in the top eigenvectors. We demonstrate that this leads to an unexpected phenomenon that longer hashing code does not necessarily yield better performance. In this work, we introduce a random subspace strategy to address this limitation. At first, a small fraction of the whole feature space is randomly sampled to train the hashing algorithms each time and only the top eigenvectors are kept to generate one piece of short code. This process will be repeated several times and then the obtained many pieces of short codes are concatenated into one piece of long code. Theoretical analysis and experiments on two benchmarks confirm the effectiveness of the proposed strategy for hashing.

Categories and Subject Descriptors

H.3.3 [**Information Systems**]: Information Search and Retrieval

General Terms

Algorithms, Experimentation, Measurement

Keywords

Image Retrieval, Random Subspace, Binary Codes, Hamming ranking

1. INTRODUCTION

With the rapid development of Internet, large scale visual databases with high dimensionality are everywhere on the Web. These huge databases pose significant challenges to visual search since the linear exhaustive search is really time consuming. To address this issue, many hashing based methods for approximation nearest neighbor (ANN) search have been proposed recently [6, 1, 7, 2, 3]. In these approaches, hash functions are learned to map the nearby points in the original space into similar binary codes. Searching with binary codes will be very fast because the Hamming distance between them can be efficiently calculated in the modern CPU.

As one main branch of the existing hashing methods, the eigenvalue decomposition based approaches [8, 7, 2, 9] have attracted much attention. Spectral Hashing (SH) [8] treats the hashing problem as spectral embedding problem and calculates the bits by thresholding a subset of eigenvectors of the graph Laplacian. Anchor Graph Hashing (AGH) [7] follows the same idea of SH but utilizes anchor graph to obtain tractable low-rank adjacency matrices. PCAH [2] simply generates linear hash functions with PCA projections, which are the eigenvectors of the data covariance matrix.

For these eigenvalue decomposition based methods, typically the variances of different projected dimensions are different and thus the information caught by different dimensions is unbalanced. In general, the top eigenvectors carry most of the information (variance) while the remainders are usually less informative or even noisy. This will result in an unexpected phenomenon that longer hashing codes do not necessarily yield better performance. As highlighted in Figure 1, when the code length exceeds 8, increasing number of bits leads to poorer mean average precision (MAP) performance on MNIST with both PCAH and AGH. Some recent work such as Iterative Quantization (ITQ) [2] have been proposed to overcome this problem, but there still lack of theoretical guarantee that longer codes will give better result than that of the shorter ones. Furthermore, the dimensionality of visual data is generally very high and this is the main difficulty widely encountered in many eigenvalue decomposition based methods, *e.g.*, the time complexity of PCA is $O(nd^2 + d^3)$ where n is the size of training set and d is the dimensionality of the data.

In this work, we attempt to leverage the random subspace strategy to deal with these problems mentioned above for binary codes learning. We aim to concatenate many pieces of short codes generated with the eigenvalue decomposition based method into one piece of long code with the expectation that the longer code will be "stronger". However, it is clear that if the many pieces of short codes are identical, the obtained long code won't catch more information and will yield the same retrieval result as the short code. Inspired from random decision forests [4], we adopt the ran-

SIGIR'14, July 6–11, 2014, Gold Coast, Queensland, Australia.

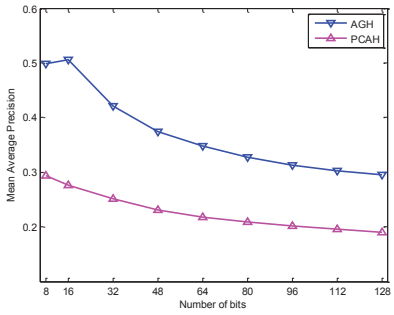


Figure 1: Mean Average Precision (MAP) of PCAH and AGH with various bits on MNIST.

dom subspace technique to generate diverse short codes. In particular, each time we randomly sample a low dimensional subspace of the training data to feed the base hashing learner, *e.g.* PCAH or AGH, and only the top eigenvectors are kept to generate one piece of short code for all the data. The subspace dimensionality is smaller than in the original feature space, while the number of training objects remains the same. This process will be repeated several times and then the obtained many pieces of short codes are concatenated into one piece of long code. We can theoretically prove that the longer codes tend to have a better result than the shorter ones under such a strategy. Since only a subspace of the original feature space is employed each time and the whole processes can be completed in parallel with independent computation units, our strategy is beneficial to computation complexity. Extensive experiments on two large scale datasets demonstrate that the proposed methods can significantly outperform the state of the art hashing methods.

2. PROPOSED STRATEGY FOR HASHING

This section gives details and theoretical analysis about the proposed random subspace strategy for hash function learning. Let $\mathcal{X} = \{x_1, x_2, \cdots, x_n\}$ denote a set of n data points, where $x_i \in \mathbb{R}^d$ is the ith data point. We denote $X = [x_1, x_2, \cdots, x_n] \in \mathbb{R}^{d \times n}$ as the data matrix. The binary code matrix of these points is $H = [h_1, h_2, \cdots, h_r] \in \{-1, 1\}^{n \times r}$, where r is the code length. Hashing code for one point is a row of H and denoted as $code(x_i)$.

2.1 Random Subspace for Hashing

The Random Subspace Method (RSM) is a combining technique proposed by Ho [4] for classification tasks. Many weak classifiers are conducted in random subspaces of the data feature space and combined by simple majority voting in the final decision. The most well known application of RSM is the Random Decision Forests [4]. In the past decades, RSM has already been proved to be an efficient way to improve the performance of weak classifiers.

For the eigenvalue decomposition based hashing methods, the amount of information (variance) caught in different eigenvectors differs significantly and the most discriminative information is often contained in the top eigenvectors. The short codes generated with only the top eigenvectors often give better ranking performance than the longer ones in these methods. In spite of this, the data representation capability of short codes is limited, and the most recently proposed hashing methods [3, 2] often use relatively longer code to achieve better performance.

Algorithm 1 Random Subspace for Hashing

Input: A base hashing learner, a training set $\mathcal{X} = \{x_1, x_2, \cdots, x_n\}$, the number of sampled features p, the number of piece of short codes K, the code size of short code t.

Output: n hashing codes of $K \times t$ bits as well as $K \times t$ hash functions.

1: Generate K replicates $\{\mathcal{X}^{(i)}\}_{i=1}^{K}$. Each replicate $\mathcal{X}^{(i)}$ contains p features randomly selected from the whole d dimensional feature space.
2: **for** i = 1, $\cdots$, K **do**
3: Train t hash functions $h_1^{(i)}, h_2^{(i)}, \cdots, h_t^{(i)}$ with $\mathcal{X}^{(i)}$.
4: **Coding:** for j = 1, $\cdots$, n, do
5: $code^{(i)}(x_j) \longleftarrow [h_1^{(i)}(x_j), h_2^{(i)}(x_j), \cdots, h_t^{(i)}(x_j)]$
6: **end for**
7: Concatenate the K pieces of short codes $\{code^{(i)}\}_{i=1}^{K}$ of each sample into one piece of $(K \times t)$ bits binary code $[code^{(1)}, code^{(2)}, \cdots, code^{(K)}]$.

In this paper, we introduce the random subspace strategy designed for classification problems to binary codes learning. In the first step, a base hashing learner, *e.g.* PCAH or AGH, is applied to generate short binary codes, i.e., only the top eigenvectors are employed. In order to generate different pieces of short codes, we randomly select $p < d$ features from the d-dimensional training set $\mathcal{X}$ each time, and this modified training set $\mathcal{X}^{(i)}$ is fed to the base hashing learner. With this process repeated several times, we can obtain multiple pieces of short but discriminative binary codes for each data. Then we concatenate these short codes into a piece of long code, which can be proved to be equivalent to the combination (voting) step in RSM. The proposed strategy for hashing can be summarized as in Algorithm 1. To the best of our knowledge, it is the first time that random subspace strategy is introduced to binary codes learning.

Since only a subspace of the original feature space is employed each time, the time complexity will be less than the base hashing learners. In addition, an important benefit of random subspace strategy for hashing is that it is inherently favorable to parallel computing. With this benefit, although the learning process has to be repeated K times, they can be completed in parallel with K computation units.

2.2 Theoretical Analysis

The essence of hashing is to find a set of binary codes for items in the database so that two similar points in the original space will have a small Hamming distance in the Hamming space, while two dissimilar points will have a large Hamming distance. The Hamming distance $d(x, y)$ between $code(x)$ and $code(y)$ can be directly converted to a similarity measure $s(x, y)$ in Hamming space:

$$s(x, y) = \frac{r - d(x, y)}{r} \tag{1}$$

where r is the length of code, $d(x, y) \in [0, r]$, $s(x, y) \in [0, 1]$, and smaller Hamming distance corresponds to larger similarity. From this perspective, we can consider that the essence of hashing is to find a set of binary codes for the data so that the similarities evaluated with these codes can match the ground truth. Here we denote the ground truth similarity between sample x and y as $f(x, y) \in [0, 1]$.

Lemma 1: *Concatenating K pieces of short codes of t bits into one piece of $K \times t$ bits long code, then the similarity between two samples evaluated with the long code is the mean of those evaluated with the K pieces of short codes.*
Proof: Let's denote the Hamming distance and similarity between two samples evaluated with different short codes as $d_i(x,y)$ and $s_i(x,y)$, where $i = 1, 2, \cdots, K$. Denote those evaluated with the long code (obtained by concatenating short codes) as $d_l(x,y)$ and $s_l(x,y)$. With Eq.(1) we have

$$\begin{aligned} s_l(x,y) &= \frac{K \times t - d_l(x,y)}{K \times t} \\ &= \frac{1}{K}\left(\frac{K \times t - \sum_{i=1}^{K} d_i(x,y)}{t}\right) \\ &= \frac{1}{K}\left(\sum_{i=1}^{K}\left(\frac{t - d_i(x,y)}{t}\right)\right) \\ &= \frac{1}{K}\sum_{i=1}^{K} s_i(x,y) \end{aligned}$$

Theorem 1: *With Algorithm 1, the similarity between two samples evaluated with the long code (obtained by concatenating short codes) tend to be closer to the ground truth than that evaluated with the short code.*
Proof: As shown in Lemma 1, the aggregated similarity evaluated with long code is:

$$s_l(x,y) = \mathbf{E}[s_i(x,y)] \tag{2}$$

With simple algebra

$$\begin{aligned} \mathbf{E}[f(x,y) - s_i(x,y)]^2 = f^2(x,y) - 2f(x,y)\mathbf{E}[s_i(x,y)] \\ + \mathbf{E}[s_i^2(x,y)] \end{aligned} \tag{3}$$

Since $\mathbf{E}[Z^2] \geq (\mathbf{E}[Z])^2$, we have

$$\mathbf{E}[s_i^2(x,y)] \geq (\mathbf{E}[s_i(x,y)])^2 \tag{4}$$

Eq.(3) can derive

$$\mathbf{E}[f(x,y) - s_i(x,y)]^2 \geq (f(x,y) - \mathbf{E}[s_i(x,y)])^2 \tag{5}$$

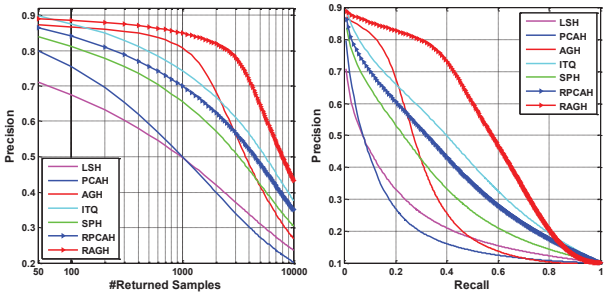
Plugging in Eq.(2), we arrive

$$(f(x,y) - s_l(x,y))^2 \leq \mathbf{E}[f(x,y) - s_i(x,y)]^2 \tag{6}$$

From Eq.(6) we can get some insights on how the longer code improve the ranking performance. How much improvement we can get depends on how unequal the Eq.(4) is. The effect of diversity is clear. If $s_i(x,y)$ does not change too much with different i the two sides will be nearly equal, and the random subspace strategy will not help. The more highly variable the $s_i(x,y)$ are, the more improvement aggregation may produce. But s_l is always superior to s_i in theory.

2.3 Connection with LSH

Locality Sensitive Hashing (LSH) [6, 1] generates a batch of random projections to embed the data into Hamming space. Owing to the inner randomness, LSH based methods have nice theoretical properties. In [6], the authors have proved that two similar samples will be embedded into close codes with high probability and this probability will increase as the code size increases. Actually, as pointed out in [2], LSH is guaranteed to yield exact Euclidean neighbors in the limit of infinitely many bits.



(a) (b)

Figure 2: (a) is the Precision-Recall curve on MNIST dataset for 64 bits. (b) is the corresponding Precision curve. (Best viewed in color)

For the random subspace method, abundant theoretical studies have been established for classification tasks [4]. According to Hoeffding inequality [5], when the base classifiers are mutually independent, the generalization error of the final decision reduces exponentially to the ensemble size (number of the base classifiers), and ultimately approaches to zero as the ensemble size approaches to infinity. Similar theory can be applied in our approach, but here the generalization error is the deviation to the true similarity and the ensemble size is the the length of code. As we have proved in Theorem 1, longer codes will result in smaller deviation to the true similarity, which is the same as in LSH.

From another point of view, if we treat one piece of short code in our method as a "super" bit, our method can be seen as a special case of LSH. The difference is that in LSH the hash functions are randomly generated but in our method we introduce randomness via randomly sampling the feature space and every "super" bit here is learned with consideration of the data.

3. EXPERIMENTS

To verify the random subspace strategy for hashing, we take two state of the art hashing algorithm as base hashing learners, i.e. PCAH and AGH. Their random subspace schemes are denoted as RPCAH and RAGH.

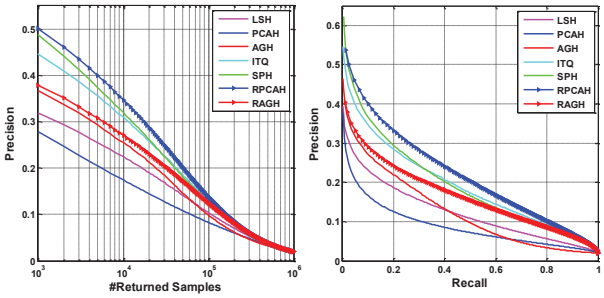
Comparison experiments are conducted on two widely used benchmarks: **MNIST**[1] and **GIST-1M**[2]. MNIST consists of 70K 28×28 grey scale handwritten digit images. GIST-1M contains 1 million GIST descriptors extracted from random images. Each descriptor is of 960 dimensions. For both dataset, we randomly select 1,000 data points as queries and use the remaining as gallery database and also training set. For MNIST, because all of the images are labeled, the ground truth is defined as semantic neighbors based on the digit labels. For GIST-1M, a returned point is considered as a true neighbor if it lies in the top 2 percentile points closest to a query. We adopt the Hamming ranking method to evaluate the performance.

To validate the effectiveness of our methods, we compare them with several state-of-the-art hashing methods including LSH [1], PCAH [2], AGH [7], ITQ [2] and Spherical Hashing (SPH) [3]. For the proposed RPCAH and RAGH, we randomly sample 70 percents of the whole feature space to learn 16 bits each time (i.e. $p = 70\% \times d$ and $t = 16$).

[1] `http://yann.lecun.com/exdb/mnist/`
[2] `http://corpus-texmex.irisa.fr/`

Table 1: The Hamming ranking performance of different algorithms with different code length on MNSIT and GIST-1M. Mean Average Precision (MAP) is reported. The best results are highlighted in bold.

Methods	MNIST				GIST-1M			
	32-bits	64-bits	96-bits	128-bits	32-bits	64-bits	96-bits	128-bits
LSH	0.2473	0.2528	0.3033	0.3472	0.0903	0.1480	0.1774	0.1853
ITQ	0.4489	0.4659	0.4722	0.4777	0.1878	0.2188	0.2307	0.2383
SPH	0.3177	0.3606	0.3795	0.3854	0.1544	0.2137	0.2447	0.2635
PCAH	0.2512	0.2181	0.2015	0.1905	0.1088	0.0914	0.0791	0.0718
RPCAH	0.3817	0.4282	0.4289	0.4536	**0.1951**	**0.2480**	**0.2736**	**0.2890**
AGH	0.4215	0.3476	0.3131	0.2945	0.1346	0.1455	0.1464	0.1460
RAGH	**0.5349**	**0.5808**	**0.5991**	**0.6044**	0.1652	0.1838	0.1904	0.2050



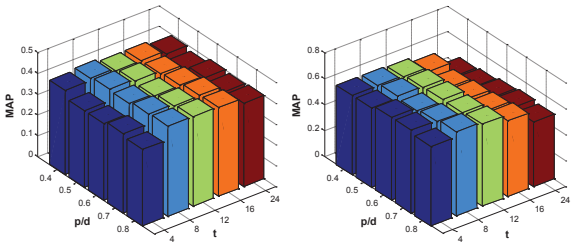
(a) (b)

Figure 3: (a) is the Precision-Recall curve on GIST-1M dataset for 64 bits. (b) is the corresponding Precision curve. (Best viewed in color)

The MAP scores of RPCAH, RAGH and other baselines are shown in Table 1. From which we can find that when the code size is small data-dependent methods like SPH and ITQ are generally better than LSH. However, as the code size increases, the MAP scores of eigenvalue decomposition based methods like PCAH and AGH decrease. In contrast, LSH results in a better performance as the code size increases. At 128 bits, the MAP score of LSH is superior to PCAH and AGH on both two datasets. This is due to the theoretical convergence guarantee of LSH. We also observe that our RPCAH and RAGH achieve a great improvement than their base methods PCAH and AGH for all code sizes on both two datasets. Besides, the performance of both RPCAH and RAGH increases as the code size increases. The improvement from 32 bits to 64 bits is prominent, and becomes stable when the code size exceeds 96 bits. This verifies the claims we made in the previous section, and is a major characteristic of our methods.

Figure 2(a)(b) and Figure 3(a)(b) show the Precision curves and Precision Recall curves on 64 bits with different methods on MNIST and GIST-1M, respectively. From the results we can also see that our proposed methods have remarkable improvement than the state of art methods. In Figure 2(a), the precision decreases in all methods as the number of retrieved points increases, but our methods decrease more slowly. In Figure 2(b) RAGH has a precision of 74% when the recall is 0.4, and the most competitive method is ITQ which only arrives at 50%. In Figure 3(b) RPCAH has a precision of 24% as the recall is 0.4 while the most competitive SPH only arrives at 21%.

We also study the sensitiveness of parameters p and t of RPCAH and RAGH. Due to the space limit, we only report the results on MNIST. The performance variations with different parameters are shown in Figure 4. From these results we observe that the proposed strategy is not sensitive to the sampling rate (p/d) as it is in a reasonable scope. This allows us to sample a small fraction (*e.g.* 40%) of the whole feature space to train the hash functions every time, which can accelerate the whole training process.



(a) RPCAH on MNIST with different p and t (b) RAGH on MNIST with different p and t

Figure 4: Parameter Sensitivity Analysis

4. CONCLUSION

We proposed a random subspace scheme for binary codes learning. The key idea was concatenating many pieces of diverse short codes generated via random subspace strategy into one piece of long code. Extensive experiments on real datasets demonstrated the effectiveness of our approach.

5. ACKNOWLEDGEMENTS

This work was supported in part by 973 Program (Grant No. 2010CB327905), National Natural Science Foundation of China (Grant No. 61170127, 61332016).

6. REFERENCES

[1] M. Charikar. Similarity estimation techniques from rounding algorithm. In *ACM symposium on Theory of computing*, pages 380–388, 2002.

[2] Y. Gong and S. Lazebnik. Iterative quantization: A procrustean approach to learning binary codes. In *CVPR*, 2011.

[3] J. Heo, Y. Lee, J. He, S. Chang, and S. Yoon. Spherical hashing. In *CVPR*, 2012.

[4] T. K. Ho. The random subspace method for constructing decision forests. *TPAMI*, 20(8):832–844, 1998.

[5] W. Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

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

[7] W. Liu, J. Wang, S. Kumar, and S. Chang. Hashing with graphs. In *ICML*, 2011.

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

[9] D. Zhang, J. Wang, D. Cai, and J. Lu. Self-taught hashing for fast similarity search. In *SIGIR*, 2010.