The 10th International Conference on
Computer Science & Education (ICCSE 2015)
July 22-24, 2015. Fitzwilliam College, Cambridge University, UK ThA3.1

An Improved Algorithm for Locality-Sensitive Hashing

Wei Cen
School of Information Science and Engineering
Xiamen University
Xiamen, China
cenweipr@163.com

Kehua Miao
School of Information Science and Engineering
Xiamen University
Xiamen, China
zxkd@xmu.edu.cn

Abstract*—We present an improved Locality-Sensitive Hashing for similarity search under high dimension search. Our scheme improves the running time based on the earlier algorithm Locality-Sensitive Hashing for hamming distance and euclidean distance. In this paper we have collected a database of The MNIST DATABASE, we proposed nearest neighbor search in the database and can get a good result in an acceptable time. The experimental results show that our data structure is up to about 10 times faster than ordinary Locality-Sensitive Hashing when working on a database of 60000 samples. At the same time, the accuracy rate and recall rate are higher than earlier algorithms.

***Index Terms*—Locality-Sensitive Hashing, high dimension, nearest neighbor, data structure.**

I. INTRODUCTION

A similar search problem is aimed at collection of objects (e.g., documents, images) that are characterized by a collection of relevant features and represented as points in a high-dimension attribute space. Given queries in the form of points in the space, we are required to find the nearest (most similar) objects to the queries. The particular interesting and well-studied case is d-dimensional Euclidean space. The problem is of major importance to a variety of applications; for example: data compression [2], databases and data mining, information retrieval, image and video databases, data analysis. In these areas, the problem we face usually are vast amounts of data and of high dimensions, which we also call "curse of dimensionality". For example, in information retrieval for text documents, the dimension of every vector point is up to several thousands, so how to quickly and accurately find the most similar to a querying point become a difficulty. Principal Components Analysis (also named as Latent Semantic Indexing) performs dramatic improvements over dimension-reduction techniques which can reduce the dimension to mere a few hundreds.

In the traditional indexing methods, kd-tree [15], SR-Tree [16] can return exact results when the dimension is very low (say, for d equal to 2 or 3), but as the dimension increase, their performances decrease quickly [9]. In1999, P. Indyk and R. Motwani proposed a Locality-Sensitive Hashing (LSH) based on hamming distance [1]. The idea is that transforms the coordinates of the vector points in high-dimensional data to positive integers. And then convert the positive integers into binary code (0 or 1) so as to get the hamming distance between vector points. In 2004, the paper [11] proposed Locality-Sensitive Hashing using p-Stable distributions, this method makes vector points directly applied in high dimension coordinate. In this paper, the resulting query time bound is free of large factors and is simple and easy to implement. Its experiments show that its data structure is up to 40 times faster than kd-tree. At the same time, there are many other papers [4,13,12,14] about improvement of it. Locality-Sensitive Hashing plays an important role in many areas like image search [8], find duplicate pages, audio retrieval, text processing. As the increase of data and dimension of data, we need to find some things similar in a short time. In this paper, we improve the LSH algorithm and make the query time greatly reduced.

We support our conclusion by empirical evidence. We performs experiments on handwritten digit data base which is of extensive research in image processing. It contains 60000 of images from 0 to 9, where each image represented as a point in d-dimension space, for d up to 784. The results shows that our scheme is significantly faster than earlier methods with higher accuracy rate and recall rate.

The rest of the paper is organized as follows. Section II introduces the nearest neighbor search. Section III introduces Locality-Sensitive Hashing. Section IV describes our scheme. Section V, the experiments section. Section VI is the conclusion of this paper.

II. NEAREST NEIGHBOR SEARCH

In similarity search, objects are characterized by a collection of relevant features and are represented as points in a high dimension space. When given a collection of points and a distance function such as hamming distance function or Euclidean distance function between them, nearest search is one of the most important forms in similarity search. Nearest neighbor search has been applied in image processing [6], local match feature [7] and so on. Its definition as follows.

Definition 1 (Nearest Neighbor Search (NNS))

Give a set of P of n objects represented as points in a normed space l_d^p, preprocess P so as to efficiently answer queries by finding the point in P closest to a queries by finding the point in P closest to a query point q.

The definition generalizes naturally to the case where we want to return K points. Specifically, in the K-Nearest

Neighbors Search (K-NNS), we wish to return K points in the database that closest to the query point.

III. Locality-Sensitive Hashing

An important technique [1] to solve NN problem is locality sensitive hashing or LSH. Its idea is to connect the possibility of collision in the same bucket between points p and point q with their distance. That is, if point p and point q are close to each other, then there is greater probability they collision; but instead if the greater distance between p and q, the more smaller probability they collision.

Definition 2. An LSH family need to satisfy the two conditions as follows:

- If $d(q, v) \leq d_1$, then the probability $h(q) = h(v)$ is at least p_1.
- If $d(q, v) > d_2$, then the probability $h(q) = h(v)$ is at most p_2.

$d(q, v)$ denotes the distance between q and v, $d_1 < d_2$, $h(q)$ and $h(v)$ represents the hash value of q and v. If hash functions satisfy the two conditions above, we can call it (d_1, d_2, p_1, p_2)-sensitive. We can get one or more hash tables when hashing for data through one or more hash functions of (d_1, d_2, p_1, p_2)-sensitive, this process is know as Locality-Sensitive hashing.

A. The creation and querying models of Locality-Sensitive hashing

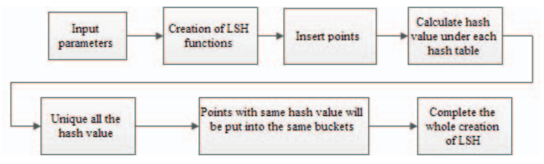
Input parameters
Creation of LSH functions
Insert points
Calculate hash value under each hash table
Unique all the hash value
Points with same hash value will be put into the same buckets
Complete the whole creation of LSH

Fig. 1. Creation of Locality-Sensitive hashing

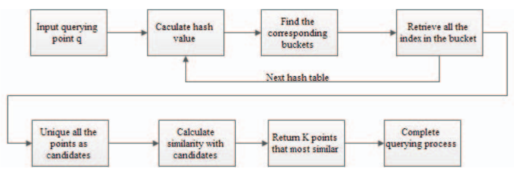
Input querying point q
Caculate hash value
Find the corresponding buckets
Retrieve all the index in the bucket
Next hash table
Unique all the points as candidates
Calculate similarity with candidates
Return K points that most similar
Complete querying process

Fig. 2. Querying model of Locality-Sensitive hashing

B. Selection of parameter K and parameter L

According to the principal of Locality-Sensitive hashing, the number of hash keys K and number of hash tables L is very important. So in order to make Locality-Sensitive hashing has a good performance on query search. We need to select appropriate K and L that must ensure the follow two conditions under a constant probability.

- If exist $v^* \in B(q, r_1)$, then for j = 1,2,...L. There has $g_j(v^*) = g_j(q)$;
- The total number of collisions with q is less than 3L, that is

$$\sum_{j=1}^{L}\left|\left(P - B(q, r_2)\right) \cap \left(g_j^{-1}\left(g_j(q)\right)\right)\right| \leq 3L. \tag{1}$$

IV. Our Improved LSH Scheme

In this section we present our algorithm in detail.

After LSH have created which uses earlier method, we can execute the query search. The step of our improved LSH algorithm implements query search when input a querying points q as shown below.

1) For every circle j = 1,2,...,L.

a) Calculate hash value for q under different hash table , according to hash value find the corresponding buckets.

b) Retrieving the index numbers in the buckets.

c) Save index numbers in iNN which we call them as candidates.

2) After circle above is completed.

a) Sort those index numbers in iNN as the frequency of occurrence.

b) Take out top 2L points with high frequency.

c) Calculate similarity with q according distance function, and return K points that most similar.

V. Experiments

The data we use in this experiment is handwritten digit recognition [5], data in it include digit (see Fig.3) from 0 to 9 and the size of every picture is 28*28 pixels, more than 6000 different pictures in average. We can see each image as a vector point, every vector points with integers from 0 to 255. we create the LSH data structure for 60000 vector points and 10000 points for testing query time. In this experiment, we mainly comparing our improved method with LSH based on hamming distance and LSH based on euclidean distance (E2LSH).

Programming language we use is Matlab, running platform is Windows 7 on the machine with Intel core i3-4010U CPU 1.7GHz and 6.0G memory.

Our detail step to do experiments as follows:

a) We need to transform every image (see Fig.3) to the a format with row vector, it means that an image is a row vector .

b) Building LSH structure for 60000 points, we set (k=20, L=24), it means we create 24 hash tables, 20 hash keys.

c) Plot graph (see Fig.4), it shows that as the k increase, the accurate rate increase from this graph we find that when k is larger than 20, accurate nearly the same. So set k=20 is appropriate, which k=20 means choose the top 20 with high frequency as the candidates.

d) Get the accuracy rate and recall rate when comparing our results with LSH based on hamming distance and euclidean distance, the results are shown in Fig.5, Fig.6, Fig.7, Fig.8.

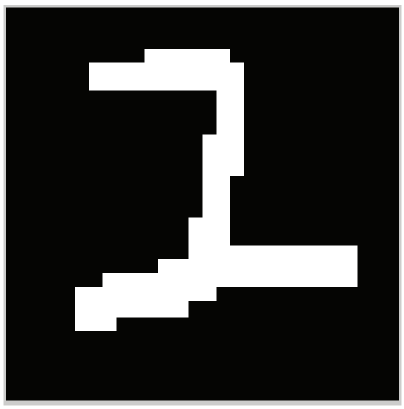

Fig. 3. Example of digit image

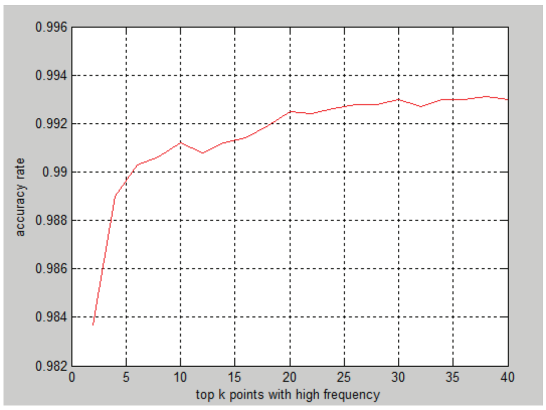

Fig. 4. Accuracy rate for different k

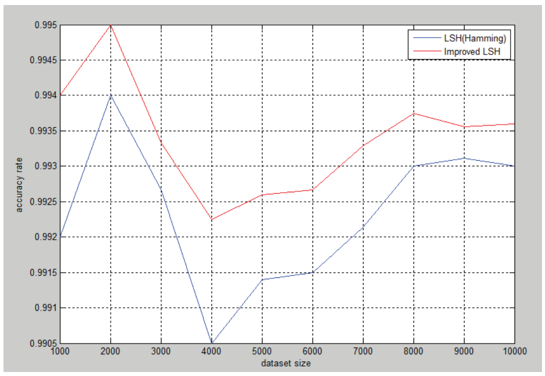

Fig. 5. Accuracy rate for LSH based on hamming distance and our improved algorithm

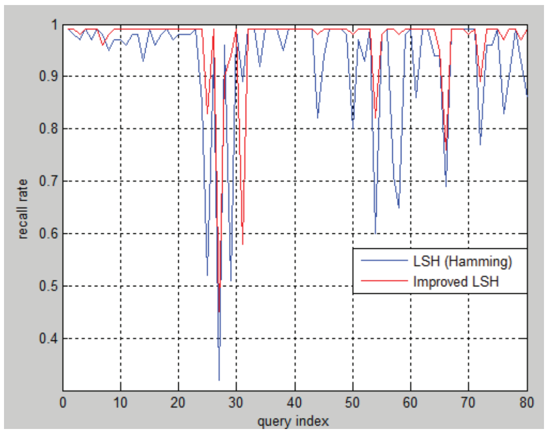

Fig. 6. Recall rate for LSH based on hamming distance and our improved algorithm

Fig.5 and Fig.6 shows the comparing of LSH which is based on hamming distance with our improved LSH method. Fig.5 shows the change of accuracy rate as the query points increase. In order to compare recall rate of the two methods, we random get 80 different points in the database , Fig.6 shows the recall rate of each query point. The results we can see that our improved algorithm has a higher accuracy rate and recall rate than LSH which based on hamming distance.

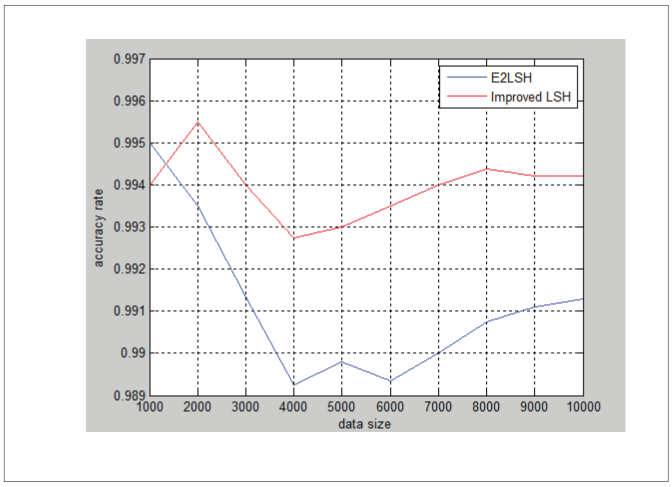

Fig. 7. Accuracy rate for E2LSH and our improved algorithm.

Fig.7 and Fig.8 tell us the comparing of accuracy rate and recall rate between E2LSH and our improved algorithm. The two results shows that our proved algorithm has a better performance over E2LSH.

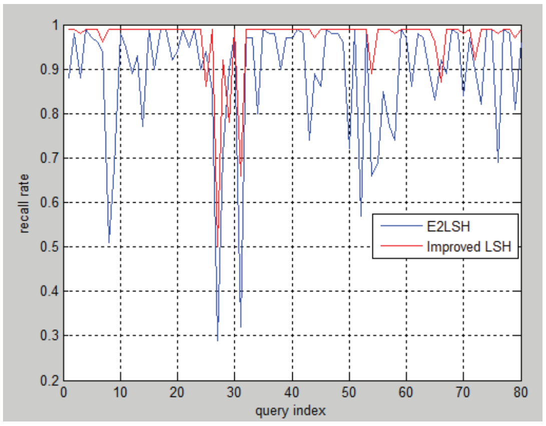

Fig. 8. Recall rate for E2LSH and our improved algorithm

After we create the hash tables, we random get 1000 points from the data and for every points we make a query. Table I is the hit rate we get, the result shows that hit rate of improved LSH is higher than other two methods. Table II is the average query time for every point in querying. It tells us our improved LSH algorithm is about 10 times faster than other two methods.

TABLE I. THE ACCURACY RATE OF THREE SOLUTIONS

Hit Rate	**Three Methods**		
	LSH (Hamming)	***E2LSH***	***Improved LSH***
Hit Rate	99.30%	99.15%	99.35%

TABLE II. THE QUERY TIME OF THREE SOLUTIONS

	Comparing Three Methods		
	LSH (Hamming)	***E2LSH***	***Improved LSH***
Query Time(seconds)	0.50	0.30	0.034

VI. CONCLUSION

Based on the handwritten digits, our improved LSH algorithm can significantly shorten time for every query point 0.034 seconds, about 10 times faster than other methods. At the same time when compared with LSH based on hamming distance and euclidean distance, our improved method have a higher accuracy rate and recall rate than other two methods.

REFERENCES

[1] P. Indyk and R. Motwani. Approximate Nearest Neighbor: Towards Removing the Curse of Dimensionality. In Proceeding of the 30th Symposium on Theory of Computing.1998.pp.604-613.

[2] A. Gersho and R.M.Gray. Vector Quantization and Data Compression.Kluwer,1991.

[3] H. Hotelling Analysis of a complex of statistical variables into principal components. Journal of Educational Psychology, 27(1933).pp.417-441.

[4] P. Indyk and R. Motwani. Similarity Search in High Dimensions via Hashing Proceedings of the 25th VLDB conference,Edinburgh,Scotland,1999.

[5] Y . Le Cunn. The mnist database of handwritten digits. Available at http://yann.lecun.com/exdb/mnist/.

[6] M.Raginsky and S. Lazebnik. Locality-sensitive binary codes from shift-invariant kernels. NIPS,2009.

[7] C.Strecha, A. Bronstein, M. Bronstein and P.Fua. Ldahash: improved matching with smaller descriptors. IEEE PAMI,34:66-78,2012.

[8] Darun Tang,Beining Huang,Rongfeng Li, Wenxin Li. A Person Retrieval Solution Using Finger Vein. 2010 International Conference on Pattern Recognition.

[9] Webber R, Schek H J,Blott S. A Quantitative Analysis and Performance Study for similarity Search Methods in High Dimensional Spaces[C]. In Proceedings of the 24th VLDB Conference,1998:194-205.

[10] Taher H. Haveliwala, Aristides Gionis and Piotr Indyk. Scalable Techniques for Clustering the Web. WebDB Workshop,2000.

[11] M. Datar,N. Immorlica, P.Indyk, and V. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. Proceedings of the ACM Symposium on Computational Geometry,2004.

[12] Kulis B, Grauman K. Kernelized locality-sensitive hashing[J]. Pattern Analysis and Machine Intelligence, IEEE Transactions on,2012,34(6):1092-1104.

[13] By Alexandr Andoni and Piotr Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimesions. Communications of the ACM January2008/Vol.51.No.1.

[14] Locality-sensitive hashing using stable distributions. page 55 - page 67.

[15] J.L.Bentley. K-D trees for semi-dynamic point sets. In Proc. Of the 6th ACM Symposium on Computational Geometry(SCG).1990. 187-197.

[16] N.Katayama and S.Satoh. The SR-tree: An index structure for high dimensional nearest neighbor queries. In Proc.of ACM Intl.Conf.on Management of Data(SIGMOD).1997.369-380.