ENTROPY BASED LOCALITY SENSITIVE HASHING

Qiang Wang, Zhiyuan Guo, Gang Liu, Jun Guo

Beijing University of Posts and Telecommunications
Email:wangqiang086373@gmail.com

ABSTRACT

Nearest neighbor problem has recently been a research focus, especially on large amounts of data. Locality sensitive hashing (LSH) scheme based on *p*-stable distributions is a good solution to the approximate nearest neighbor (ANN) problem, but points are always mapped to a poor distribution. This paper proposes a set of new hash mapping functions based on entropy for LSH. Using our new hash functions the distribution of mapped values will be approximately uniform, which is the maximum entropy distribution. This paper also provides a method on how these parameters should be adjusted to get better performance. Experimental results show that the proposed method will be more accurate with the same time consuming.

Index Terms— Locality sensitive hashing (LSH), approximate nearest neighbor (ANN), entropy, information retrieval, large-scale

1. INTRODUCTION

The problem of high-dimensional nearest neighbor search arises in a large variety of database applications and it has recently been a research focus. Index-based locality sensitive hashing (LSH) is the most effective method, owing to its sub-linear search time and controllable success probability.

The theory of LSH was originally proposed by P. Indyk and R. Motwani in 1998 [1]. A. Gionis and P. Indyk implemented a hash-based approximate nearest neighbor search in hamming space a year later [2]. In 2004, P. Indy presented LSH scheme based on *p*-stable distributions in original non-hamming space [3]. In the same year, in the field of natural language processing Deepak Ravichandran improved LSH, mapping points to binary vectors, and applied it to large-scale noun clustering [4]. A. Andoni and P. Indyk divided the space into lattice in LSH [5] in 2006. In the next year, multi-probe LSH was proposed, which not only returned points in original buckets, but also points in nearby buckets [6]. In 2008, Wei Dong provided guidance on how to adjust parameters automatically to optimize the program [7]. In 2010, Loic Pauleve and Herve Jegou introduced clustering into LSH, leading to the divided space conforming to the data distribution [8].

The key insight behind LSH is that hash functions should map adjacent points to the same buckets with higher probability, but map points far from each other to the same buckets with lower probability. We always need to construct several hash tables to increase the collision probability due to the low collision probability of neighbor points mapped to the same bucket by one hash table. However, one of the key problems when using traditional hash functions is the lack of adaptation of data after mapping. The distribution of mapped values is always uneven. Some buckets own many points, but others own few points. If points in buckets can be uniformly distributed, we can get almost the same number of candidates for each query. The average time for candidates calculating will be reduced and the total retrieval time will be less.

Considering the uniform distribution of points in buckets, this paper presents a set of new hash functions based on maximum entropy. Firstly we mark the cut-off positions using the hash functions. Then every point is mapped to an integer according to the cut-off positions. In our method, by choosing reasonable cut-off positions the points in different buckets will be nearly uniformly distributed. The paper also proposes a method of optimizing parameters of the new hash functions. The experimental results show that the proposed method can achieve the same accuracy using less time compared with original LSH.

The paper is organized as follows: Section 2 describes the LSH scheme based on *p*-stable distributions. The new hash function and parameter adaptation method are proposed in Section 3. We finally report experimental results and conclusions in Section 4 and Section 5 respectively.

2. E2LSH

The key idea of LSH is that neighbor points are more likely mapped to the same bucket but points far from each other

This work was partially supported by National Natural Science Foundation of China under Grant No. 61175011 and 61171193, the 111 project under Grant No.B08004, Innovation Fund of Information and Communication Engineering School of BUPT in 2011, and the Next-Generation Broadband Wireless Mobile Communications Network Technology Key Project under Grant No. 2011ZX03002-005-01.

are more likely mapped to different buckets. To implement LSH, for a high-dimensional space S^d with distance measure D, the LSH family is defined as follows:

A family $H=\{h: S^d \rightarrow U^1\}$ is called (r_1, r_2, p_1, p_2)-sensitive *for D if for any* $\boldsymbol{v}, \boldsymbol{q} \in S^d$*:*

If $\boldsymbol{v} \in B(\boldsymbol{q}, r_1)$*, then* $\mathrm{P_{rH}}[h(\boldsymbol{p})=h(\boldsymbol{v})] \geq p_1$,

If $\boldsymbol{v} \notin B(\boldsymbol{q}, r_2)$*, then* $\mathrm{P_{rH}}[h(\boldsymbol{p})=h(\boldsymbol{v})] \leq p_2$.

We then randomly choose k h-functions from the family H to form a function family $G=\{g: S^d \rightarrow U^k\}$ such that $g(\boldsymbol{v})=(h_1(\boldsymbol{v}),h_2(\boldsymbol{v}),\ldots,h_k(\boldsymbol{v}))$, where $h_i \in H$. For any $\boldsymbol{v} \in S^d$, the point will be mapped to a k-dimensional vector with a g-function. The vector is the hash value and can be regarded as a bucket. In fact one g-function can only give a low collision probability of neighbor points, that is to say neighbor points are unlikely mapped to the same bucket in one hash table. By choosing L g-functions $g_1, g_2,\ldots,g_L$ from G at random, we can build L hash tables to increase the collision probability. One of the critical factors for LSH is the construction of hash function family H. The function should ensure neighbors are mapped to the same hash value, and far points are mapped to different values. LSH hash function family based on p-stable distributions can well meet the requirements above.

P-stable distribution is defined as follows:

A distribution C is called p-stable, if there exists $p \geq 0$ such that for any n real numbers $v_1,\ldots,v_n$ and variables $\boldsymbol{X_1},\ldots,\boldsymbol{X_n}$ with distribution C, the random variable $\sum_i v_i \boldsymbol{X_i}$ has the same distribution as the variable $(\sum_i | v_i |^p)^{1/p}\boldsymbol{X}$, where $\boldsymbol{X}$ is a random variable with distribution C.

In Euclidean space, the standard normal distribution is a typical 2-stable distribution. For any $\boldsymbol{v} \in S^d$ and variable $\boldsymbol{a}$ with distribution $N^d(0, E)$ (E is the unit matrix, so each dimension of $\boldsymbol{a}$ is independent standard normal distribution), the projection $\boldsymbol{a} \cdot \boldsymbol{v}$ is distributed according to the normal distribution $N(0, \| \boldsymbol{v} \|)$, where $\| \boldsymbol{v} \|$ is the Euclidean distance of $\boldsymbol{v}$ from the origin. For any $\boldsymbol{v}, \boldsymbol{q} \in S^d$ and variable $\boldsymbol{a}$ with distribution $N^d(0, E)$, the distribution of the projection distance $(\boldsymbol{a} \cdot \boldsymbol{v} - \boldsymbol{a} \cdot \boldsymbol{q})$ is $N(0, \| \boldsymbol{v}-\boldsymbol{q} \|)$. So the projection distance is consistent with original Euclidean distance in probability and we can build hash functions based on p-stable property. Hash function $h_{\boldsymbol{a},b}$ is given by:

$$h_{\boldsymbol{a},b}(\boldsymbol{v}) = \lfloor (\boldsymbol{a} \cdot \boldsymbol{v} + b)/w \rfloor \quad (1)$$

In the formula, $\boldsymbol{a}$ is a random vector with distribution $N^d(0, E)$, $\boldsymbol{v}$ is a point in S^d, w is the quantization step, and b is a real number chosen uniformly from the rang $[0, w]$. The g-function is composed of k $h_{\boldsymbol{a},b}(\boldsymbol{v})$, namely $g(\boldsymbol{v})=(h_1(\boldsymbol{v}),h_2(\boldsymbol{v}),\ldots,h_k(\boldsymbol{v}))$. Every point is stored in a bucket with the projection value of a g-function as the hash key. We can construct L g-functions and build L hash tables to increase the number of collision points. The collision probability is $1-(1-p^k)^L$, in which p is the collision probability of one $h_{\boldsymbol{a},b}(\boldsymbol{v})$ function, and p can be calculated as follows:

$$p(c) = \Pr_{\boldsymbol{a},b}[h_{\boldsymbol{a},b}(\boldsymbol{v}) = h_{\boldsymbol{a},b}(\boldsymbol{q})] = \int_0^w \frac{1}{c} f_p\left(\frac{t}{c}\right)\left(1-\frac{t}{w}\right)dt \quad (2)$$

where $\boldsymbol{v}$ and $\boldsymbol{q}$ are points chosen randomly from S^d, $c=\| \boldsymbol{v}-\boldsymbol{q} \|$, f_p is the probability density function (in Euclidean space, f_p is the standard normal probability density function), and w is the quantization step (always set to 4). According to the two formulas above and the accuracy input by the user, we can obtain the relationship between k and L. Finally we can get the optimal value of k and L by minimizing the hash computation time and candidate calculation time.

LSH based on p-stable distributions speeds up neighbor searching greatly, but when choosing hash functions it does not consider the actual data distribution and only chooses hash functions from family H randomly. In fact, the distribution of data is not uniform, resulting in some hash functions mapping to concentrated values and others mapping to discrete values. There is a big difference between the numbers of points in different buckets, leading to diverse collision probability. If we can make the projection uniform and the number of points in different buckets almost the same, then the number of candidates for each query point is little variable. The average retrieval time will be reduced and LSH will be improved. Taking into account uniform projection, this paper proposes an entropy based LSH.

3. ENTROPY BASED LOCALITY SENSITIVE HASHING

3.1. Projection

To improve LSH, most methods are query-directed. Multi-probe LSH expands the number of query buckets to increase candidates [6]. Query-adaptive LSH chooses buckets of efficient hash functions by calculating hash values of query points [9]. When building the index if we can design a set of new hash functions to optimize the mapped values, the performance will be better.

In the theory of LSH based on p-stable distributions, different mapped regions always have different number of points after the projection of h-functions. When it extends to the k-dimensional vector the difference will be greater. Figure 1 gives a mapped result of part of the audio dataset in Section 4 using an h-function. It can be seen from Figure 1 that the mapped values are distributed roughly according to Gaussian distribution. There are more points near the origin, resulting in uneven distribution. The original points are mapped to k-dimensional vectors by k h-functions, which will be concentrated in the vicinity of the zero vector. So the distribution of k-dimensional vectors is uneven by far and the number of points in each bucket is different extremely. When querying, if the mapped vector of the query point is near the zero vector, a large number of candidates will be compared, leading to much time consumed on candidate comparison. If the mapped vector is far from the zero vector, fewer or no candidates will lead to a lower accuracy. If we

can use an appropriate mapping method to make *k*-dimensional mapped vectors uniform and the distribution entropy maximum, the number of candidates for each query will be nearly the same. It could be a good solution to these problems, reducing search time and increasing search accuracy.

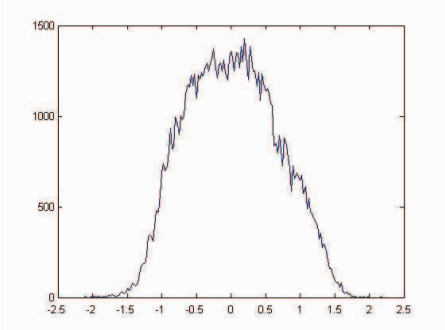

Figure.1. Example of mapping results of an *h*-function.

3.2. Entropy based Locality Sensitive Hashing Algorithm

Based on the description of the problem above, the paper presents a set of new hash functions and explains it in Euclidean space. For a high-dimensional space S^d and a data set of *N* points, $\boldsymbol{a}$ is a random vector with the distribution $N^d(0,E)$ and $\boldsymbol{v}$ is a point chosen randomly from S^d, the initial mapped value can be calculated by the following formula:

$$H'_{\boldsymbol{a}}(\boldsymbol{v})=\boldsymbol{a}\cdot\boldsymbol{v} \tag{3}$$

The mapped set is $\{h'_1,h'_2,\ldots,h'_N\}$. We then introduce another parameter *r*, which represents the number of quantization. The mapped value will be quantified to (0, 1,…,*r*-1). The process is as follows:

First, sort initial mapped values and record $1/r$, $2/r$,…, $(r-1)/r$ cut-off points as $q_1,q_2,\ldots,q_{r-1}$. The number of points will be the same in each mapped region segmented by cut-off points. Then make use of the following formula to obtain the quantization value:

$$H_{\boldsymbol{a}}(\boldsymbol{v})=\begin{cases}0 & if \quad H'_{\boldsymbol{a}}(\boldsymbol{v})\le q_1\\ 1 & if \quad H'_{\boldsymbol{a}}(\boldsymbol{v})> q_1 \quad and \quad H'_{\boldsymbol{a}}(\boldsymbol{v})\le q_2\\ \vdots & \\ r\text{-}2 & if \quad H'_{\boldsymbol{a}}(\boldsymbol{v})> q_{r-2} \quad and \quad H'_{\boldsymbol{a}}(\boldsymbol{v})\le q_{r-1}\\ r\text{-}1 & if \quad H'_{\boldsymbol{a}}(\boldsymbol{v})> q_{r-1}\end{cases} \tag{4}$$

The formula is the new *h*-function. We can then randomly select *k* functions from the family *H* to form *g*-function. The mapped *k*-dimensional vector is the final hash value, regarded as a bucket. It can be seen that points are mapped to *r* regions evenly by *h*-functions. After the projection of a *g*-function composed of *k* *h*-functions, the distribution of points is almost uniform. The paper introduces distribution entropy to evaluate hash functions. The formula is as follows:

$$E(g)=-\sum p(N(i))*log(p(N(i))) \tag{5}$$

$N(i)$ is the number in the bucket *i*, and $p(N(i))$ is the collision probability of the bucket *i* ($p(N(i))= N(i)/N$). The entropy of mapped values can be calculated for a *g*-function by the formula. For a constant number of buckets, uniform collision can result in larger entropy. So the uniformization of collision probability is actually to maximize the distribution entropy.

In the original LSH, we should create *L* hash tables. We also choose *k*L* *h*-functions, that is to say *L* *g*-functions, to build *L* hash tables.

3.3. Optimization of parameters

The proposed hash function is different from original LSH based on *p*-stable distributions, so *k* and *L* should be optimized differently. Besides we introduce another parameter *r* and the adjustment of *r* is more important to get best performance. The following will show how to optimize parameters.

The LSH search time can be decomposed into two terms. The first term is $T_g=O(dkL)$ for computing the *k*L* *h*-functions. The second term is $T_c=O(d*\#collisions)$ for computing the distance to all the candidates encountered in the searched buckets. *#collisions* is the number of candidates in all the buckets.

To calculate T_g, *k* and *L* should satisfy the following relationship:

$$1-(1-p_r^k)^L\ge 1-\delta \tag{6}$$

δ is the accuracy input by users, p_r is the collision probability of neighbor points when an *h*-function maps points to integers. Since p_r has a relationship with *r*, the number of neighbor points to return and the data set, we can estimate each p_r by a small part of real data set and the number of points needed to return. The formula implies the relationship between *k* and *L* is $L\ge(log\delta)/log(1-p_r^k)$, and we choose $L=\lceil(log\delta)/log(1-p_r^k)\rceil$.

To calculate T_c, we need know the expected value of *#collisions*. The expected value is $E[\#collisions]$ and $E[\#collisions] = L*N/r^k$, because the collision probability of points in each bucket is distributed roughly according to uniform distribution.

The optimization of *k*, *L* and *r* is defined as follows:

Obj: $Min(T_g+T_c)$

St: 1) $T_g=kL*t_g$

2) $T_c=(L*N/r^k)*t_c$

3) $L=\lceil(log\delta)/log(1-p_r^k)\rceil$

where t_g is the time of computing an *h*-function and t_c is the time of computing distance to a candidate. In the definition t_g and t_c are relevant to the dimension of the data and the computer. *r* can change from 2 to 6, *k* is set from 2 to 60, *L* is determined by *k*, p_r and δ. We can adjust *r* and *k* to minimize (T_g+T_c) and obtain the optimal *r*, *k* and *L*.

4. EXPERIMENTS

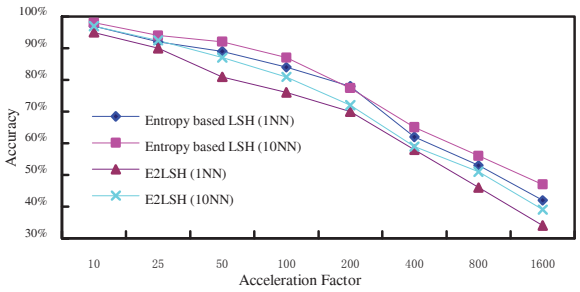

Figure.2. The results of 39-dimensional MFCC points.

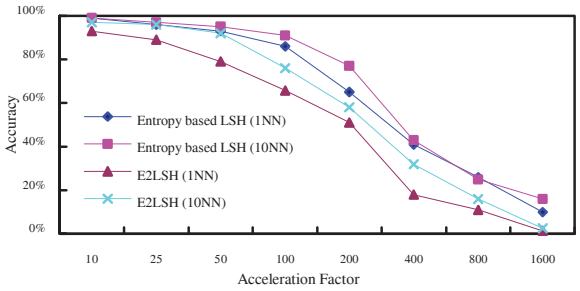

Figure.3. The results of local feature descriptor points.

To demonstrate the proposed method, tests are performed on two corpora. One is a dataset of 1,001,000 39-dimensional MFCC feature points extracted from an audio database [10] [11]. The other dataset is 11,000 128-dimensional local feature descriptor points extracted from an image database [12]. In the two datasets, one million or ten thousand points are used to build LSH index and one thousand points are queries. The evaluation criteria are the accuracy and acceleration factor for 1NN and 10NN. The accuracy is the ratio of right number returned by LSH and the total number of neighbor points demanded to return. The acceleration factor is the ratio of linear searching time and LSH time. The results may slightly fluctuate for each test because of the randomness of generated hash functions. So for the best *r, k*, and *L*, we build LSH index 10 times and compare the average performance.

Figure 2 and 3 show the searching results of 39-dimensional MFCC points and 128-dimensional descriptor points. It can be seen that the entropy based LSH is superior to the original E2LSH. With the same acceleration factor, the accuracy of 1NN is increased up to 8.2% for the MFCC dataset and up to 23.5% for the image dataset. The accuracy of 10NN is increased up to 6.2% and 19.1% respectively. We can also see that LSH can get better performance for the larger dataset.

5. CONCLUSIONS

The paper presented a new hash function scheme, which can make the distribution of mapped values uniform. The collision probability will be almost even in each bucket and the accuracy will be higher with the same time consuming. To get better performance we can also use multi-probe or query-adaptive strategy for the proposed new hash functions. The paper also proposed a method to automatically optimize parameters, but the estimation of parameter p_r may be inaccurate sometimes, leading to parameter tuning failure. The next work is to further optimize parameters, especially parameter p_r. If we can give a formula to calculate p_r, the parameter tuning will be more precise.

6. REFERENCE

[1] Piotr Indyk, Rajeev Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," in *Proc. the thirtieth Annual ACM Symposium on Theory of Computing*, 1998.

[2] Aristides Gionis, Piotr Indyk, Rajeev Motwani, "Similarity search in high dimensions via hashing," in *Proc. the 25th International Conference on Very Large Data Bases (VLDB)*, 1999.

[3] Mayur Datar, Piotr Indyk, Nicole Immorlica, Vahab S. Mirrokni, "Locality sensitive hashing scheme based on *p*-stable distributions," in *Proc. Annual Symposium on Computational Geometry (SOCG)*, June 9–11, 2004.

[4] Deepak Ravichandran, Patrick Pantel, Eduard Hovy, "Randomized algorithms and NLP: using locality sensitive hash function for high speed noun clustering," in *Proc. the 43th Annual Meeting on Association for Computational Linguistics*, 2005.

[5] Alexandr Andoni, Piotr Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," in *Proc. the 47th Annual Symposium on Foundations of Computer Science* (*FOCS'06*).

[6] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li. "Multi-probe LSH: efficient indexing for high-dimensional similarity search," in *Proc. the 24th International Conference on Very Large Data Bases (VLDB)*, 2007.

[7] Wei Dong, Zhe Wang, William Josephson, Moses Charikar, Kai Li, "Modeling LSH for performance tuning," in *Proc. CIKM'08,* October 26–30, 2008.

[8] Loic Pauleve, Herve Jegou, Laurent Amsaleg, "Locality sensitive hashing: a comparison of hash function types and querying mechanisms," *Pattern Recognition Letters,* 2010.

[9] Herve Jegou, Laurent Amsaleg, Cordelia Schmid and Patrick Gros, "Query-adaptative locality sensitive hashing," in *Proc. International Conference on Acoustics, Speech, and Signal Processing (ICASSP)*, 2008.

[10] Logan B, "Mel Frequency Cepstral Coefficients for Music Modeling," in *Proc. the International Symposium on Music Information Retrieval (ISMIR)*, 2000.

[11] Qiang Wang, Gang Liu, Zhiyuan Guo, Jun Guo, "Structural fingerprint based hierarchical filtering in song identification," in *Proc. International Conference on Multimedia & Expo (ICME)*, 2011.

[12] K. Mikolajczyk, T. Tuytelaars, C. Schmid, "A comparison of affine region detectors," *IJCV,* 2005, pp. 43-72.