RESEARCH ARTICLE OPEN ACCESS

A Survey of Locality-Sensitive Hashing

Yang Yang LI
(School of Information Science and Technology, Jinan University, Guangzhou, China)

----------**********----------

Abstract:

Nearest neighbor problem is very common. Linear algorithms can work well when dealing with low-dimensional datasets,however, it is inefficient for massive and high-dimensional datasets. Hence, how to quickly find one or more point thatclosest to the requiredpoint from a high-dimensional dataset,has become a difficult problem.Therefore, the Locality-Sensitive Hashing (LSH) was proposed to resolve such issues. To better understand the LSH algorithm, this paper gives a comprehensive analysis and summarizes on LSH.Firstly,it introduces the original LSH algorithm. And then the P-stable LSH algorithm are carefully analysed.Finally, some improvements based on the P-stable LSH algorithm are summarized, including Multi-probe LSH andStep-Wise Probing.

Keywords **—ANN, Locality-Sensitive Hashing, stable distribution, Multi - Probe LSH.**

----------**********----------

I. INTRODUCTION

LSH was first introduced in 1998 by Indyk and Motwani [1] to solve the Approximate Nearest Neighbor (ANN) problem. The concept of LSH was first proposed embedding in Hamming code, so it was only applied in Hamming space, while the original data was transformed at the beginning, has caused the datato lose some information invisibly. In 2004, an algorithm based on the stable distribution for the $\boldsymbol{L_p}$ norm ($\boldsymbol{0 < p <= 2}$) was designed by Datar et al. [2], which has greatly expanded the space of LSH. In the last decade, many people had improved the P-stable LSH. Panigrahy [3] proposed an entropy-based LSH. LV et al. [4] inspired by Panigrahy to propose a multi-probe LSH algorithm. The multi-probe LSH has a significant improvement in space and time efficiency over the original LSH. In order to balance the contradiction between query cost and query quality in LSH algorithm, Tao et al. [5] proposed a local sensitive B-tree (LSB-tree) access method.Gan et al. [6] used single LSH functions to construct a "dynamic" composite hash function and defined a new LSH scheme called collision count LSH (C2LSH), which is superior to LSB-forest in high-dimensional space.To optimize I/O efficiency, Yin et al. [7] group small hash buckets into a single bucket by dynamically increasing the number of hash functions during index construction. Huang et al. [8] proposed a query-aware LSH scheme called QALSH for c-ANN search on external storage.

II. ORIGINAL LSH

The original LSH needs to map the points to a d-dimensional hamming space H^d first. The mapping method is as shown in Equation 2-1 and 2-2.

$$p \to p' \epsilon P \quad (2\text{-}1)$$

$$p' = Unary_c(x_1)Unary_c(x_2)\cdots Unary_c(x_d) \quad (2\text{-}2)$$

Where $Unary_c(x_i)$ denotes a numeric string consisting of 0 and 1, containing x_i zeros and $c - x_i$ ones, for example:$Unary_6(2) = 110000$. This method can ensure that the distance before and after the mapping is unchanged. ie: $\forall p, q \in P, d_1(p,q) = d_H(p,q)$. Where d_1 is the Euclidean distance defined under the l_1 norm on space$P$, and d_H is the Hamming distance defined in the H^d space. In this way, the problem of finding the K points closest to the query point q in P is transformed into the K-nearest neighbor problem in the Hamming space.

Definition1 (Hash function family): *Define* $I = \{1,2,\dots,d\}$ *and a positive integer* l*, take* x *subsets from* I*, and mark these subsets as* $I_1,\dots,I_i,\dots,I_l$*. The definition* $g_i = p|I_i$ *is the projection of the vector* p *on the coordinate set* I*, that is, each coordinate in the coordinate set* I *is a position index, the bit value corresponding to the position of the vector* p *is taken, and the results are concatenated, and finally the projection is placed in the bucket.*

Definition2(Query): *For a given query point* q*, Calculate:* $g_1(q),\dots,g_i(q),\dots,g_l(q)$*. Take all the points in the hash bucket corresponding to* $g_i(q)$ *(ie all* p *points that satisfy* $g_i(q) = g_i(p)$*) as candidate sets.*

A. P-stable LSH

P-stable LSH is an evolutionary version of LSH. The problem they solve is the same, whileused in different

application method and application environment.P-stable LSH can be applied in the Euclidean space under the d-dimensional l_p paradigm, which is a huge improvement over the original LSH. The mathematical basis of P-stable LSH is the P-stable distribution. It is necessary to understand the P -stable distribution first.

Definition3 (P-stable distribution):*For a distribution D on a real set R , if there is a number p>0, for any n real numbers* $v_1, v_2, \dots, v_n$ *and* n *variables* $x_1, x_2, \dots, x_n$, *random variables* $\sum_i v_i x_i$ *and* $(\sum_i |v_i|^p)^{1/p} X$ *have the same distribution, and where X is a random variable subject to the D distribution.The distribution satisfying the above conditions is a P-stable distribution.*

The P-stable distribution is not a specific distribution, but a distribution family that satisfies a certain condition.when P=1, it represents the standard Cauchy distribution, and the density function is: $c(x) = \frac{1}{\pi}\frac{1}{(1+x^2)}$.When P = 2, it represents the standard normal distribution (Gaussian distribution) and the density function is: $g(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$.These are the two most common of the P-stable distributions.

An important application of the P -stable distribution: estimating the length $||v||_p$ of a given vector$v$in the Euclidean space$l_p$paradigm. The method is for the given d-dimensional vector $v = \{v_1, v_2, \dots, v_d\}$,Extracting d random variables from the P -stable distribution to form a d-dimensional vector $a = \{x_1, x_2, \dots, x_d\}$,Calculate the dot product $a \cdot v$ of a and v, according to the definition of P-stable distribution, there are $a \cdot v = \sum_i v_i X_i$,Therefore $a \cdot v$ and $||v||_p X$ are equally distributed.

In P-stable LSH, the property of $||v||_p$ length can be estimated by $a \cdot v$ to construct a hash function.This family of hash functions is locally sensitive. If $a \cdot v$ can estimate the length of $||v||_p$, then $a \cdot v_1 - a \cdot v_2 = a \cdot (v_1 - v_2)$ (where is the mapped distance) to estimate the length of $||v_1 - v_2||_p$ (where $||v_1 - v_2||_p$is the original distance). When the original distance $||v_1 - v_2||_p$ is small, the mapped distance is also small, so using a dot product to generate a hash function family can maintain local sensitivity. The hash function is shown as 3-1.

$$h_{a,b}(v) = \left\lfloor \frac{a \cdot v + b}{r} \right\rfloor \quad (3\text{-}1)$$

Note:

- b is a random number in $(0, r)$;
- r is the length of the segment on the straight line;
- The functions in the hash family are indexed according to the difference between a and b.

A hash function is randomly selected from the hash function family. The probability that the two vectors$v_1$and v_2 collide after mapping under the hash function as shown in 3-2.

$$p(c) = Pr_{a,b}[h_{a,b}(v_1) = h_{a,b}(v_2)] = \int_0^r \frac{1}{c} f_p\left(\frac{t}{c}\right)\left(1 - \frac{t}{r}\right) dt \quad (3\text{-}2)$$

Note:

- $c = ||v_1 - v_2||_p$ *($a \cdot v_1 - a \cdot v_2$ and cX are distributed)*
- This formula can also be thought of as a function that is only related to the value of c . The larger the value of c , the smaller the value of the function (the lower the probability of collision); the smaller the value of c, *the* larger the value of the function (the higher the probability of collision).

B. P-stable LSH similarity search algorithm

Section A introduces the construction of the P-stable distributed LSH function family. The next question is how to concretely implement the construction of the hash table and how to query the nearest neighbor.

The process of building a hash table is the process of performing a hash operation on each vector. One of the most important problems in the LSH similarity search algorithm is false negatives. One solution to reduce the false negative rate is to perform hash operations on vectors using multiple hash functions. For example, for any vector v_i, prepare k hash functions, $h_1(), h_2(), \dots, h_k()$, which are randomly selected from the LSH hash function family, so that k hash values can be obtained by calculation:$h_1(v_i), h_2(v_i), \dots, h_k(v_i)$, For the query q, using the same$k$hash functions, we can also get a set of values: $h_1(q), h_2(q), \dots, h_k(q)$. As long as one of the two sets of values is equal, v_i is considered to be a neighbor of the query q.

Through the above improvements, it has reduced the false negative rate indeed, while accompanied with increasing false positive rate. Therefore, a measure must be added based on the above method. From the LSH function family, $l$sets of functions are randomly selected, and each function group is composed of k randomly selected functions. Now the l sets of functions deal with the data separately. As long as one set is completely equal, the two data are considered to be similar.

The above is an AND then OR logic, which can be called the (k, L) algorithm.Now suppose $P = Pr[h_{a,b}(v_1) = h_{a,b}(v_2)]$, then the probability that two pieces of data are considered neighbors is: $1 - (1 - P^k)^L$.

When building a hash table, if you use a set of function values as identifiers for the hash table, there are two disadvantages: 1) Large spatial complexity; 2) Not easy to find. To solve these two problems, two hash functions are designed: H_1、H_2, see formula 3-3, 3-4 for details.

$$H_1: Z^k \to \{0,1,2,\dots,size-1\} \quad (3\text{-}3)$$

$$H_2: Z^k \to \{0,1,2,\dots,C\}. \quad C = 2^{32} - 5 \quad (3\text{-}4)$$

$size$ represents the length of the hash table, C is a large prime number. The specific algorithm for these two functions as shown in 3-5 and 3-6, where $r_i, r_i'$are two random integers:

$$H_1(x_1, \dots, x_k) = \left(\left(\sum_{i=1}^k r_i x_i\right) \bmod C\right) mod\ size \quad (3\text{-}5)$$

$$H_2(x_1, \dots, x_k) = \left(\sum_{i=1}^k r_i' x_i\right) mod\ C \quad (3\text{-}6)$$

The result of the H_1 calculation is called the "fingerprint" of a data vector; and H_2 is equivalent to the index of the

fingerprint of the data vector in the hash table. This algorithm is consistent with the basic hash table algorithm.

C. Algorithm step

The steps to solve the nearest neighbor problem using the LSH algorithm are as follows:

- The L group hash function is randomly selected from the designed LSH hash function family. Each group consists of K hash functions, which are recorded as: $\{g_1(\cdot), g_1(\cdot), \dots, g_i(\cdot)\}$, where: $g_i(\cdot) = g_i(h_1(q), h_2(q), \dots, h_k(q))$.
- Each vector is mapped to an integer vector via $g_i(\cdot)$,which is written as: $(x_1, \dots, x_k)$
- The integer vector $(x_1, \dots, x_k)$ generated in 2), calculated by H_1 , H_2 to obtain two values: index (bucket number index), fp (fingerprint);
- If there are data vectors with the same data fingerprint, they will necessarily be mapped to the same hash bucket.

III. MULTI - PROBE LSH

Multi-Probe LSH was proposed by Lv et al. in the paper "MultiProbe LSH: Efficient Indexing for High Dimensional Similarity Search". The basic LSH method needs to query many hash tables to ensure the search quality. Multi-Probe LSH is proposed to solve this shortcoming, whose core idea is to use a carefully selected probe sequence to detect multiple buckets that may contain nearest neighbor. According to the nature of the LSH, if the neighbor p of the query point q is not mapped to the bucket where p is located, then p is likely to be in the bucket near the bucket where $p$is located. The purpose of the Multi-Probe LSH is to find these adjacent buckets and increase the probability of finding q neighbors.

A. Algorithm summary

Given a query point q, the basic LSH is to query in the hash bucket $g(q) = h_1(q), h_2(q), \dots, h_k(q)$. In the Multi-Probe LSH, we define a perturbation sequence $\Delta = \delta_1, \delta_2, \dots, \delta_k$ for the query point q, and the perturbation sequence Δ can be used to query the bucket $g(q) + \Delta$. Assuming that the LSH function $H(v_1) = \left\lfloor \frac{a \cdot v_1 + b}{r} \right\rfloor$ based on the P-stable distribution is used, if r is large enough, the adjacent data points will fall on the same or adjacent values with a high probability, so the range of$q$can be constrained to be $\{-1, 0, 1\}$. Each perturbation vector acts directly on the hash value of the query object. Therefore, compared with the entropy-based local sensitive hash, it'll be more efficient to calculate the hash point of the disturbance point. Design a set of perturbation sequences to act on the set of hash values so that the number of times a hash bucket is queried is not more than once.

B. Step-Wise Probingexploration

An n -step perturbation sequence x has n non-zero coordinates, and this perturbation sequence corresponds to a hash bucket. According to the nature of the locally sensitive hash function, a one-step hash bucket is more likely to contain a neighbor of the query point than a two-step hash bucket. So firstcheck the bucket at one step, then the bucket at the second step until you find enough hash buckets. In most cases, a Step-Wise Probing within two steps is enough to find most buckets with a higher probability of success. For a locally sensitive hash algorithm with L hash tables and k local sensitive hash functions, the total number of query buckets in n steps is $L \times C_m^n \times 2^n$, and the total number of query buckets in the step is $L \times \sum_{n-1}^{s} C_m^n \times 2^n$.

C. Success probability assessment

In Step-Wise Probing, all hash values of query point q are treated equally, that is, the probability that each hash value is perturbed is the same, and the probability of adding 1 or minus 1 for each hash value is the same. However, a more optimized method of constructing a search sequence needs to consider how the individual hash values of the query points are calculated. Each hash function x maps the query point $q$to a straight line, which is divided into different regions from left to right with $r$as the width. The obtained hash value is the region to be mapped by the point q, one and the point. The point p adjacent to the $q$phase may fall in the same or adjacent area as the point$q$. In fact, the point p is to the right or left of the point q, depending on the distance of the point q from the boundary of the area where it falls.

$F_i(q) = a_1 \cdot q + b_1$ is the mapping of the query point q on the line after using the i_{th} hash function, $h_i(q)$ is the area to which the query point q is hashed, and for $\delta \in \{-1, 1\}$, definition $x_i(\delta)$is the distance from the query point q to the boundary of the region $h_i(q) + \delta$, $x_i(-1) = f_i(q) - h_i(q) \times w, x_i(1) = w - x_i(-1)$, definition $x_i(0) = 0$ for convenience. For any fixed point p, $f_i(p) - f_i(q)$ is a random variable with a Gaussian distribution with a mean of zero. The variance of this random variable is proportional to the distance between point p and point q, assuming that a sufficiently larger is chosen, which makes us interested that the point of the point falls on the region $f_i(q), f_i(q) - 1, f_i(q) + 1$ with a high enough probability, The probability density function of the Gaussian random variable is $e^{-x^2/2\delta^2}$, so the probability that the point p falls in the region $h_i(q) + \delta$ can be estimated as formula 4-2:

$$Pr\,[h(p) = h(p) + \delta] \approx e^{-Cx_i(\delta)^2} \quad (4\text{-}2)$$

Where C is a constant and its value depends on $||p - q||_2$.

The probability of success of a perturbed vector x is show in formula 4-3:

$$Pr[g(p) = g(q) + \Delta] = \prod_{i=1}^{m} \Pr[h(p) = h(p) + \delta] \approx \prod_{i=1}^{m} e^{-Cx_i(\delta)^2} = e^{-C\sum_{i=1}^{m} x_i(\delta)^2} \quad (4\text{-}3)$$

D. Query-oriented search sequence

A preliminary way to generate a perturbation sequence is to generate all the sequences, calculate their scores and sort them. However, there are $L \times (2K-1)$ perturbation sequences, and we only want to use exactly the part with the smallest score, so generating all the perturbation sequences is extremely wasteful. Some researchers have described a more efficient way to generate perturbation sequences in ascending order of scores.

First, note that the score of the perturbation vector δ depends only on the non-zero coordinates in those δ, and we expect that the perturbation vector with a low score contains only some non-zero coordinates. In perturbation vector generation, we represent the non-zero coordinates of the vector as a set of $(i,\ \delta_i)$ pairs, and an$(i,\ \delta_i)$ pair adds δ to the i_{th} hash of the query point q. Given a query point q and a hash function $h_i(i = 1,2,\dots,m)$corresponding to a hash table, corresponding to a single hash table. First calculate $x_i(\delta) = (-1,1),\ i = 1,2,\dots,m$, we arrange the $2m$ elements from large to small and define z_j as the jth element j in the sequence. Define π_j, if $z_j = x_j(\delta)$, then $\pi_j = (i,\ \delta)$, which means that the value $x_j(\delta)$ is the value that is the j_{th} smallest in the sequence. It is worth noting that $x_i(1) + x_i(-1) = w$, if $\pi_j = (i,\ \delta)$, then there must be $\pi_{2m+1-j} = (i, -\delta)$.

The perturbation vector is now represented as a subset of the set $\{1,2,\dots,2m\}$, which is equivalent to a perturbation set. For each such perturbed subset A , the corresponding perturbation vector is, the perturbation coordinates of the $\{\pi_j | j \in A\}$ are obtained from the set $\{\{\pi_j | j \in \{1,2,\dots,2m\}\}$ according to the perturbation set A and its value. The score a defining each perturbation set A is the sum of the squares of the z values of the set a, which is equal to the score of its corresponding perturbation vector.The score score(A) defining each perturbation set A is the sum of the squares of the z_j values of the set A, which is equal to the score of its corresponding perturbation vector.Thus, given that a sorted (i, δ_i) pair is stored in queue π , and the value $z_j, j = 1,\dots,2m$, the problem of generating a perturbation vector is reduced to a perturbation set from small to large scores from set $\{1,2,\dots,2m\}$.To complete this task, we need to define two operations:

$shift(A)$: This operation will replace the element $max\ (A)$ in the set A with the element $1 + ma\,x(A)$, for example: $shift(\{1,3,4\}) = \{1,3,5\}$

$expand(A)$: This operation expands the set A and adds the element $1 + ma\,x(A)$ to the set A , for example: $expand(\{1,3,4\}) = \{1,3,4,5\}$.

Two facts about this operation are important to ensure the correctness of the process of generating the disturbance set:

- For any perturbation set A, the scores of sets $shift(A)$ and $expand(A)$ are both higher than those of A.
- For any perturbation setA, there is a single $shift(A)$ and $expand(A)$ process starting from {1} until generating A

Based on the above two facts, it is easy to derive the following two properties:

- The generation process produces all valid perturbations from small to large by the set score.
- At any time, the number of elements in the heap is always one more than the number of elements before a round of operations.

IV. CONCLUSIONS

This paper introduces the original LSH algorithm to the multi-probe LSH algorithm. Through the introduction can find that LSH is accessible for all the problems that need to find the nearest neighbor,and the use of LSH can effectively improve the speed of searching. Currently, it's generally used for finding similar web pages, image retrieval, and music retrieval. However, it should be noted that the use of LSH must ensure that the data is sensitive to distance, which requires the extraction of feature tasks.

REFERENCES

[1] Indyk P, Motwani R. Approximate nearest neighbors: towards removing the curse of dimensionality[C]//Proceedings of the thirtieth annual ACM symposium on Theory of computing. ACM, 1998: 604-613.

[2] M. Datar, P. Indyk, N. Immorlica, and V. S. Mirrokni.Locality-sensitive hashing scheme based on p-stabledistributions. In SoCG, pages 253–262, 2004.

[3] Panigrahy R. Entropy based nearest neighbor search in high dimensions[C]//Proceedings of the seventeenth annual ACM-SIAM symposium on Discrete algorithm. Society for Industrial and Applied Mathematics, 2006: 1186-1195.

[4] Lv Q, Josephson W, Wang Z, et al. Multi-probe LSH: efficient indexing for high-dimensional similarity search[C]//Proceedings of the 33rd international conference on Very large data bases. VLDB Endowment, 2007: 950-961

[5] Tao Y, Yi K, Sheng C, et al. Efficient and accurate nearest neighbor and closest pair search in high-dimensional space[J]. ACM Transactions on Database Systems (TODS), 2010, 35(3): 20

[6] Gan J, Feng J, Fang Q, et al. Locality-sensitive hashing scheme based on dynamic collision counting[C]//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. ACM, 2012: 541-552.

[7] Yin S, Badr M, Vodislav D. Dynamic multi-probe lsh: An i/o efficient index structure for approximate nearest neighbor search[C]//International Conference on Database and Expert Systems Applications. Springer, Berlin, Heidelberg, 2013: 48-62.

[8] Huang Q, Feng J, Zhang Y, et al. Query-aware locality-sensitive hashing for approximate nearest neighbor search[J]. Proceedings of the VLDB Endowment, 2015, 9(1): 1-12.