Solving Approximate Nearest Neighbor by Hash Aggregate and Clustering in High Dimensional Euclidean Space

Hangfei Hu1, Hongya Wang1*, Zongyuan Tan1

1 School of Computer Science and Technology, Donghua University, Shanghai, 201620, China

*Corresponding author’s e-mail: hywang@dhu.edu.cn

Abstract. Approximate Nearest Neighbor search has attracted great attention in the field of information search. The method based on Locality-Sensitive Hashing has good performance with probability guarantee, which needs m hash functions for collision detection. In the hash table corresponding to each hash function, collision counts are performed respectively during querying, which will result in additional collision counts and longer response time. To avoid the shortcoming, we aggregate the m hash values as the representation of the original data object. Then the aggregate values of all data objects in the dataset are clustered. In the process of querying, it is needed to find the class of the aggregate value of the query for searching. Experiments show the effectiveness of our method.

1. Introduction

With the continuous progress of science and technology, massive amounts of data are generated every day, and the dimension of data is also higher and higher. How to effectively index data and achieve efficient retrieval has been a hot research topic of related scholars. Nearest Neighbor (NN) search is an important research direction. Given a data set, a distance function and a query, the nearest neighbor search is to find a data object in the database to minimize the distance between the data object and the query point under the specified metric distance function. Nearest neighbor search is widely used in many fields, including pattern recognition [1], statistical classification [2], computer vision, data compression, image retrieval [3] and so on. The original solution is linear scanning which traverses the entire data set to find the exact nearest neighbor of the query. The time complexity and space complexity of the algorithm are very high. Some methods based on spatial partition, such as R-tree[4], can find the exact nearest neighbor of a given query in the dataset. With the increase of data dimension, dimension disaster will occur. The query performance of these spatial partition methods is not even as good as the original linear scan.

In high-dimensional space, accurate nearest neighbor search is a huge challenge. However, in many applications, it is acceptable that the algorithm can return approximate nearest neighbor. As an alternative, c-Approximate Nearest neighbor search (c-ANN) has attracted much attention. Given a data set, a query q and an approximate proportion c, c-ANN algorithm can return the approximate nearest neighbor o of q, where o satisfies ||o, q|| ≤ c × ||o*, q||, o* is the real nearest neighbor of q. Locality-Sensitive Hashing (LSH) is one of the classical approximate nearest neighbor search algorithms [5]. The algorithm based on LSH needs m hash functions and filtering threshold l, like C2LSH [6]. In the process of querying, the algorithm counts the collision separately in each hash table, which will result in additional collision count and long response time. In order to avoid the shortcomings, we aggregate the m hash values corresponding to the data point, and the aggregation results of all data objects in the...
dataset are clustered. In the process of querying, m hash values of the query are also aggregated and the category information of the aggregation of the query is also taken into account to find the ANN for the query. Experiments show that our method has good query performance in most of the data sets.

2. Related work

ANN is a classic problem. In recent years, related scholars mainly focus on the following three aspects: algorithms based on LSH, algorithms based on product quantization (PQ) and algorithms based on graph. The algorithms based on LSH are generally probability guaranteed algorithms, which can obtain relatively stable query results in ANN retrieval. And they need to select a number of hash functions from the family of locality sensitive hash functions to build an index for all elements in the database. Each hash function corresponds to a hash table. After the query point arrives, collision detection is performed in each hash table. Then, combined with certain retrieval strategies, the ANN of the query is found. The basic ideas of the algorithms are to design more reasonable search strategies, or change the storage structure of the index, or use a more reasonable locality sensitive hash function, so as to improve the query accuracy. For example, SK-LSH changes the storage structure in order to reduce the IO and improve the query accuracy, which stores the data points close to each other in the same index file [7]. Like other classical algorithms, they also solve the nearest neighbor search problem from these perspectives, such as C2LSH [6], QALSH [8], LazyLSH algorithm [9], VHP algorithm [10].

3. Preliminary

3.1. Problem Definition

For the c-ANN problem, it is hoped that the result $o$ returned by the algorithm satisfies $|o, q| \leq c \times |o_*, q|$, where $o_*$ is the real nearest neighbor of query $q$. That is to say that the returned $o$ should satisfy $|o, q| \leq c \times r$, when $r = |o_*, q|$. In the high-dimensional Euclidean space, $B(q, R)$ is a hypersphere with $q$ as center and $R$ as radius. Under the specified approximate ratio $c$, if the data point $o$ satisfies $B(q, c \times r) = \{o \in R^d | |o, q| \leq c \times r\}$, it is reasonable to return $o$ as the ANN of the query $q$. As shown in Figure 1, it is reasonable to return one of these data points ($o_1$, $o_2$, $o_3$, $o_*$) as the approximate nearest neighbor of the query $q$. Moreover, c-ANN algorithm still has a great possibility to return the real nearest neighbor $o_*$ of the query $q$.

![Figure 1. c-nearest neighbor search results](image1)

3.2. Locality sensitive hash function

LSH algorithm is a data independent algorithm, which can find the nearest neighbor for the query with a certain probability for any data set. LSH algorithm uses several locality sensitive hash functions to index a given data set. These locality sensitive hash functions can hash the nearer data objects to the
same hash bucket or adjacent hash buckets with large probability. Next, the definition of locality sensitive hash function family is given as follows [6]:

Let \( p_1 = p(r) \) and \( p_2 = p(cr) \), for any data point \( o, q \in R^d \):
- If \( o \in B(q, r) \), the collision probability of \( o \) and \( q \) is \( Pr[h(o) = h(q)] \geq p_1 \)
- If \( o \notin B(q, cr) \), the collision probability of \( o \) and \( q \) is \( Pr[h(o) = h(q)] \leq p_2 \)

The set of hash functions that satisfy the above characteristics is \((r, cr, p_1, p_2)\)-sensitive. In high-dimensional Euclidean space, a kind of commonly used locality sensitive hash functions is defined as follows:

\[
h(o) = \hat{a} \cdot \hat{o}
\]

(1)

The dimension of \( \hat{a} \) is the same as the data dimension, and each dimension of \( \hat{a} \) is generated by the standard normal distribution \( N(0,1) \).

\[
X_i \sim N(0,1) (1 \leq i \leq n), \text{ if } \hat{a} = (X_1, X_2, ..., X_n)
\]

Taking two-dimensional space as an example, the results of data objects in the database after being processed by hash function in formula (1) are shown in the figure 2.

The following is a brief description that using the hash function of formula (1) to process data point and query point can keep the Euclidean distance information of them in the original space. If the query \( q = (v_{11}, v_{12}, ..., v_{1n}) \) and the data object \( o = (v_{21}, v_{22}, ..., v_{2n}) \), the results by hash function of formula (1) are \( h(q) = v_{11} \times X_1 + v_{12} \times X_2 + \cdots + v_{1n} \times X_n \) and \( h(o) = v_{21} \times X_1 + v_{22} \times X_2 + \cdots + v_{2n} \times X_n \). Then the hash distance between the data object \( o \) and the query point \( q \) is \( h(o) - h(q) \), and \( h(o) - h(q) = (v_{21} - v_{11}) \times X_1 + (v_{22} - v_{12}) \times X_2 + \cdots + (v_{2n} - v_{1n}) \times X_n \).

For random variable \( X \) and arbitrary constant \( c \), the mathematical expectation of \( cX \) is \( c \) times of that of \( X \), and the variance of \( cX \) is \( c^2 \) times of that of \( X \), as shown in formula (2). For two independent random variables \( X \) and \( Y \), the mathematical expectation of \( X + Y \) is equal to that of \( X \) plus that of \( Y \), and the variance of \( X + Y \) is equal to that of \( X \) that of \( Y \). Owing to \( X_i \sim N(0,1) (1 \leq i \leq n) \). \((v_{2i} - v_{1i}) \times X_i \sim N(0, (v_{2i} - v_{1i})^2)\). According to formula (3), \((v_{21} - v_{11}) \times X_1 + (v_{22} - v_{12}) \times X_2 + \cdots + (v_{2n} - v_{1n}) \times X_n \sim N(0, (v_{21} - v_{11})^2 + (v_{22} - v_{12})^2 + \cdots + (v_{2n} - v_{1n})^2) \). Let \( s = |o, q| = (v_{21} - v_{11})^2 + (v_{22} - v_{12})^2 + \cdots + (v_{2n} - v_{1n})^2 \), then the hash distance calculated by formula (1) can preserve the original Euclidean distance information of data points and the query.

4. Our method

4.1. Hash aggregation

The methods based on LSH adopt \( m \) hash functions and collision detections are carried out under each hash function separately, which will lead to additional collision counts and longer online query response time. In 3.2, we present that the hash distances between the query and the data objects can retain original Euclidean distance information. We aggregate the corresponding hash values of \( m \) hash functions to represent the original data point like SRS [11], and the basic form of the hash function is shown in the formula (1). As shown in figure 3, the data point \( o \) with dimension \( d \) is processed by \( m \) hash functions to obtain \( o' \), where \( o' = [h_1(o), h_2(o), ..., h_m(o)] \). If \( d = 3, m = 2 \), \( h_1() = [0.1, -0.1, 0.2] \), \( h_2() = [0.3, 0.1, 0.5] \) and \( o = [2, 5, 1] \), it is easy to get \( h_1(o) = 2 \times 0.1 + 5 \times (-0.1) + 1 \times 0.2 = -0.1 \), \( h_2(o) = 2 \times 0.3 + 5 \times 0.1 + 1 \times 0.5 = 1.6 \). Therefore, the result of hash aggregation is \( o' = [-0.1, 1.6] \). In the first stage of the index, all elements in the dataset are hashed and aggregated by randomly selected \( m \) hash functions. In this way, all n-dimensional data vectors are processed into m-dimensional vectors.
4.2. Cluster processing

After clustering, these data objects with the same or similar attributes will be clustered together. When looking for the nearest neighbor of a query, we only need to find the nearest neighbor in the class it belongs to. K-means algorithm is one of the most classic clustering algorithms [12]. In this paper, Euclidean distance is used to cluster the data objects. If \( o_1 = [v_{11}, v_{12}, ..., v_{1n}] \) and \( o_2 = [v_{21}, v_{22}, ..., v_{2n}] \), the Euclidean distance of \( o_1 \) and \( o_2 \) is calculated by the formula (4).

\[
\text{dist}(o_1, o_2) = \sqrt{\sum_{i=1}^{n} (v_{1i} - v_{2i})^2}
\]

With the given parameter \( k \), \( k \) cluster centers are randomly selected. According to the Euclidean distances between the data objects and these \( k \) cluster centers, every data object can find its own class. All the \( k \) cluster centers are updated according to all the points in its own class according to the formula (5), where \( \text{centroid}(o_i) \) refers to the \( i \)-th cluster center and \( k_i \) represents the number of data objects belonging to class \( i \). According to the new clustering center, the data objects are clustered again. The algorithm iteratively updates the \( k \) cluster centers and re-clusters the data objects until the cluster centers do not change or change very little.

\[
\text{centroid}(o_i) = \left[ \frac{\sum_{j=1}^{k_i} v_{j1}}{k_i}, \frac{\sum_{j=1}^{k_i} v_{j2}}{k_i}, ..., \frac{\sum_{j=1}^{k_i} v_{jn}}{k_i} \right]
\]

If K-means algorithm is directly used to cluster a large number of high-dimensional data, the clustering time will be too long. If the low dimensional vectors after hash aggregation are used to cluster, the clustering time will be greatly reduced. In the second stage of the index, the results by hash aggregation are clustered by K-means algorithm.

4.3. Query processing

In the process of querying, the query \( q \) is hashed and aggregated to obtain \( q' \). And the class to which \( q' \) belongs can be found with the minimum Euclidean distance among all the distances between \( q' \) and the cluster centers. As shown in figure 4, \( q' \) belongs to class 1 and the data points pertained to class 1 are traversed. The data object is returned as the ANN of \( q \), which has the minimum Euclidean distance with \( q \) among all the data objects in class 1 in the original space. As for \( topk \) problem, there is a case that the number of points in the class to which the query belongs is so small that the number of points that can be traversed is less than \( topk \). Facing this situation, we traverse at least two classes to solve the problem. As shown in figure 4, we traverse the class 1, but \( topk \) problem cannot be solved. Then the class 2 should be traversed, because the distance between the cluster center of class 2 and \( q' \) is the
second smallest. The classes are constantly interviewed according to the distances between \( q' \) and the cluster centers until the number of access points exceeds \( \text{top}_k \) after accessing the current class.

5. Experiment

Two datasets are used to verify the effectiveness of our method. Audio dataset contains 53387 data objects whose dimension is 192. And Sun dataset embodies 79106 data objects and each of them is 512 dimensions. Our method is implemented in python. All experiments were done on a PC with Intel Core i5-10300H@ 2.50GHz 2.50GHz CPU, 16 GB memory. In this paper, recall is used to measure the performance of the algorithm, and the calculation method of recall is shown in the formula (6), where \( R(q) \) is a set containing the true neighbor(s) of \( q \) and \( R'(q) \) is a set that contains the neighbor(s) returned by the algorithm.

\[
\text{Recall}(q) = \frac{|R'(q) \cap R(q)|}{|R(q)|}
\]  

For Audio dataset, 100 queries are randomly selected. The recalls of the 100 queries are counted and showed in the form of histogram of frequency distribution with different \( m \) and \( k \), and the results for \( \text{top}_50 \) and \( \text{top}_100 \) are shown in figure 5. With the increase of \( k \), the number of cluster centers increased gradually, which results in the decreasing of the data points in each class. In the process of query, the number of times to calculate the actual distance between the query and the data points in the class is reduced for each query, resulting in the decline of recall. The results with \( m = 50, k = 50 \) is better than that with \( m = 50, k = 100 \), as shown in Figure 5 (a) and (c) or (b) and (d). With the increase of \( m \), the results of hash aggregation can reflect more information of the original data, so the clustering effect will be better. Therefore, the results with \( m = 100, k = 50 \) is better than that with \( m = 50, k = 50 \), as shown in Figure 5 (a) and (e) or (b) and (f). From (a) and (e), the data objects with \( m = 80, k = 50 \) whose recalls are greater than 50% are more than that \( m = 50, k = 50 \).

![Figure 5](image-url)

For Sun dataset, we also select 100 queries randomly to draw frequency distribution histograms. The results of Sun dataset are not so good as those of Audio dataset owing to the higher dimension of Sun dataset, and the results are omitted due to space constraints. Basically, with the increase of \( m \) and the decrease of \( k \), the recall corresponding to the query develops in a better direction.

6. Conclusion

In this paper, we combine hash aggregation technology with clustering method to solve ANN problem for the query. Firstly, the data objects in the dataset are hashed by the randomly selected \( m \) hash
functions. And the $m$ hash values corresponding to a data object are aggregated into a low dimensional vector as the representation of the data point. Then all the low dimensional vectors are clustered by K-means algorithm. According to the category information of the query, the ANN of the query can be easily found. Experiments show the effectiveness of our method.

**Acknowledgments**

This work is supported by the Fundamental Research Funds for the Central Universities and Graduate Student Innovation Fund of Donghua University (NO: CUSF-DH-D-2020092).

**References**

[1] J. Cheng, C. Leng, J. Wu, et al. (2014) Fast and accurate image matching with cascade hashing for 3d reconstruction. In: CVPR. Columbus. pp. 1-8.

[2] M. Datar, N. Immorlica, P. Indyk, et al. (2004) Locality-sensitive hashing scheme based on p-stable distributions. In: SoCG. New York. pp. 253-262.

[3] J. Tang, Z. Li, X. Zhu. (2018) Supervised deep hashing for scalable face image retrieval. Pattern Recognition, 75: 25-32.

[4] A. Guttman. (1984) R-trees: A dynamic index structure for spatial searching. In: SIGMOD. Boston. pp. 47-57.

[5] P. Indyk, R. Motwani. (1998) Approximate nearest neighbors: towards removing the curse of dimensionality. In: STOC. Dallas. pp. 604-613.

[6] J. Gan, J. Feng, Q. Fang, et al. (2012) Locality-sensitive hashing scheme based on dynamic collision counting. In: SIGMOD. Scottsdale. pp.541-552.

[7] Y. Liu, J. Cui, Z. Huang, et al. (2014) SK-LSH: an efficient index structure for approximate nearest neighbor search. In: VLDB. Hangzhou. pp. 745-756.

[8] Q. Huang, J. Feng, Y. Zhang, et al. (2015) Query-aware locality-sensitive hashing for approximate nearest neighbor search. In: VLDB. Hawaii. pp. 1-12.

[9] Y. Zheng, Q. Guo, A.K.H. Tung, et al. (2016) LazyLSH: Approximate nearest neighbor search for multiple distance functions with a single index. In: SIGMOD. San Francisco. pp.2023-2037.

[10] K. Lu, H. Wang, et al. (2020) VHP: approximate nearest neighbor search via virtual hypersphere partitioning. In: VLDB. Tokyo. pp. 1443-1455.

[11] Y. Sun, W. Wang, J. Qin, et al. (2014) SRS: Solving c-approximate nearest neighbor queries in high dimensional Euclidean space with a tiny index. In: VLDB. Hangzhou. pp. 1-12.

[12] M. Norouzi, D.J. Fleet. (2013) Cartesian k-means. In: CVPR. Portland. pp. 3017-3024.