Finding Nearest Neighbors for Multi-Dimensional Data

Yong Shi, Marcus Judd
Department of Computer Science
Kennesaw State University
1000 Chastain Road
Kennesaw, GA 30144
yshi5@kennesaw.edu, mjudd3@kennesaw.edu

***Abstract*–Nearest Neighbor Search problem is an important research topic in data mining field. In this paper, we discuss our continuous work on finding nearest neighbors in multi-dimensional data based on our previous research work. The research work presented in this paper improves our original algorithm by analyzing the distribution of data points on each dimension**.

***Keywords*–Similarity Search, Multi-query, Data Point Weight**

I. Introduction

As one of the important research topics in the data mining field, the nearest neighbor search problem has been studied and various approaches have been proposed in different research scenarios [2], [8], [10], [11], [13], [16]. For example, [11] proposes a scheme for nearest neighbor search by hashing data points from a data set, so that for data points close to each other, the probability of collision is much higher than those that are far from each other. The authors also conducted experiments to demonstrate the accuracy and scalability of their algorithm. [13] presents a multi-step algorithm that produces the minimum number of candidates for nearest neighbor search problem. The algorithm works well for the efficiency requirements of complex high-dimensional and adaptable distance functions. [16] provides a detailed analysis of partitioning and clustering techniques for nearest neighbor search problem. The paper also discusses an alternative organization based on approximations to make the unavoidable sequential scan as fast as possible.

II. Related Work

Traditional approaches apply similarity functions such as Euclidean distance to calculate the distance between two data points in a given data set. Such approaches often have a problem called “curse of dimensionality”, since when dimensionality goes higher, the distance between two data points becomes less meaningful [9], [6], [16], [7]. There are approaches designed for partial similarities analysis [12], [4], [3], but most of them have the problem of lack of flexibility because they require fixed subset of dimensions or fixed number of dimensions as a part of the algorithm input .

In [14], we discuss the fact that in reality, we need to find nearest neighbors to multiple query points with different level of importance. We define the distance between a data point and a set of query points, taking into consideration how important each query point is, and how dimensions should be dynamically chosen for each data point. We apply our algorithm to find nearest neighbors in different subspaces of the original data space.

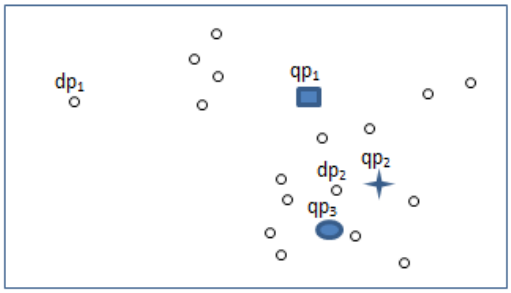

Fig. 1: A 2-dimensional data set with multiple query points where data points have different densities

III. Problem Definition

In this paper, we propose to enhance the process of the algorithm discussed in [14] by analyzing the data point distribution.

For example, figure 1 shows a 2-dimensional data set along with multiple query points. The hollow dots represent the data points. The solid square qp_1, the solid four-point star qp_2, and the solid ellipse qp_3 represent different query points. Among the data points represented by the hollow dots, the data point dp_1 is far from other data points, and the data point dp_2 is close to many data points. For many real-world applications, query points are close to dense data point area, as shown in figure 1, where qp_1, qp_2 and qp_3 are all much closer to dp_2 than to dp_1. For applications of this kind, a data point closer to other data points (such as dp_2) has a higher chance to be one of the K nearest neighbors of multiple query points, compared to dp_1. Based on this observation, we propose to enhance the algorithm in [14] by assigning a weight to each query point which represents how many neighbors it has.

In this paper we will use DS to represent the data set in our approach. DS contains data points in multi-dimensional data space. The size of DS is n and DS has d dimensions: D_1, D_2, ... D_d. Each data point in DS is a d-dimensional vector: $X_i = [x_{i1}, x_{i2}, ..., x_{id}]$ (i=1,2,...,n). The identity of X_i is i. We consider the case that there are multiple query points. Those

 ISBN: 978-1-61208-247-9

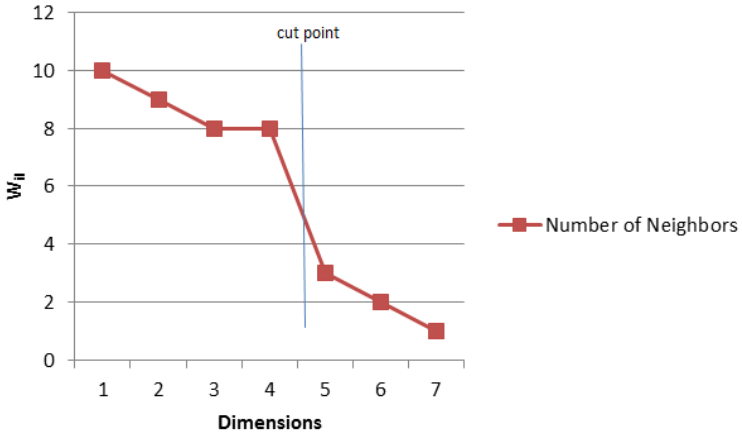

Fig. 2: Dimensions sorted by W_{il}

query points will be in the same data space as data points in DS are: $Q_j = [q_{j1}, q_{j2}, ..., q_{jd}]$. Suppose the set of the query points is Q: $Q = \{Q_1, Q_2, ..., Q_m\}$, with the size of Q being m. Both DS and Q are normalized. Each query point Q_j has a weight WQ_j that represents the importance of the distance to Q_j.

As we discussed previously, we analyze the data distribution and assign a weight W_i to each data point X_i in DS, i=1,2,...,n. The value of weight shows the density of the area X_i is in. The more neighbors X_i has, higher value W_i should have.

We first calculate the weight of X_i on each dimension. For dimension D_l, l=1,2,...,d, we calculate

$$W_{il} = |\{1 \leq j \leq n || X_{il} - X_{jl}| < \alpha, i \neq j\}| \quad (1)$$

where α is a distance threshold that determines if two data points are close to each other. If X_i is close to many data points on D_l, W_{il} will have a large value; otherwise, W_{il} will have a small value.

Once we have the weight of X_i on each dimension, we can calculate the total weight W_i of X_i in the d-dimensional data space. There are several ways to achieve that. The first solution is to calculate the average of the weights on all the dimensions:

$$W_i = \frac{\sum_{l=1}^{d} W_{il}}{d} \quad (2)$$

The second solution is to select the maximum weight from all the dimensions:

$$W_i = \max_{l=1}^{d} W_{il} \quad (3)$$

Neither of the solutions works well. If we calculate the weight as the average of all the weights, those dimensions on which X_i is not close to many neighbors will weaken the weight of X_i. If we calculate the weight as the maximum value of all the weights, we disregard a lot of useful information from most of the dimensions. To avoid the disadvantages of both solutions, we design the weight W_i of X_i as follows: first we sort the dimensions based on the value of W_{il} in the descending order, shown in figure 2. Next we find the first sharp downward part as the cut point in the second half of the ordered list. All the dimensions before the the cut point are those on which X_i has many neighbors. If there is no sharp point at all in the ordered list, we simply set the cut point as the middle position in the list.

Suppose there are h dimensions before the cut point. We calculate W_i as

$$W_i = \frac{\sum_{p=1}^{h} W_{ip}}{h} \quad (4)$$

W_i is the average of the weights from dimensions on which W_{il} is high. This definition of W_i collects the information of all the dimensions on which X_i has many neighbors.

Thus the final set of the weights for the data points is

$$\mathbf{W} = \{W_1, W_2, ..., W_d\} \quad (5)$$

where W_i i=1,2,...,d is defined in formula (4).

In the next step, we define $\Delta_{ij} = [\delta_{ij1}, \delta_{ij2}, ..., \delta_{ijd}]$ as the array of differences between X_i (i=1,2,...,n) and Q_j (j=1,2,...,m) on all the dimensions (D_1,D_2,...,D_d). δ_{ijl} is calculated as:

$$\delta_{ijl} = WQ_j * |x_{il} - q_{jl}|. \quad (6)$$

IV. Algorithm

In our algorithm, we try to find K nearest neighbors for Q, given a data set DS, a query set Q and the value of K.

We calculate K nearest neighbors for Q on each dimension. The first step is to calculate δ_{ijl} based on equation (6). We next sort the data points based on δ_{ijl} on each dimension D_l, l=1,2,...,d, for each query point Q_j, j=1,2,...,m. We then define KS_{jl} as the set which contains the ids of the first K data points in the sorted list. Let $KS'_l = KS_{1l} \cup KS_{2l} \cup ... \cup KS_{ml}$. We define KS_l as the set which contains the ids of K data points which appear most frequently in KS'_l.

The next step is to calculate the weight for each data point X_i as we discussed in equation (4). To compute the distance between a data point and the query set, we define

 ISBN: 978-1-61208-247-9

Algorithm WMQKNN (*DS: data set, Q: query point set, D: dimensions, K: number of data points required, ω: threshold for calculation of query point closeness*)
Begin
For each $X_i \in DS$ and each query point $Q_j \in Q$, calculate $\Delta_{ij} = [\delta_{ij1}, \delta_{ij2}, ..., \delta_{ijd}]$ in which $\delta_{ijl} = |x_{il} - q_{jl}|$;
Sort the data points in DS based on δ_{ijl} for each query point Q_j and each dimension D_l ;
Generate KS_{jl} which contains the first K ids in the sorted list;
Generate KS'_l as union of KS_{jl} j=1,2,...,m;
Generate KS_l which contains K ids from KS'_l with the highest frequencies;
Calculate W_i for each data point X_i;
For each data point X_i, generate $B_i = [b_{i1}, b_{i2}, ..., b_{id}]$ in which b_{il} = 1, if $i \in KS_l$; b_{il} = 0, if $i \notin KS_l$;
Generate GS as union of KS_l, l=1, 2, ..., d;
For each data point X_i, where $i \in GS$, calculate $WMSD(X_i, Q)$;
Sort GS based on $WMSD(X_i, Q)$;
Let set WMQKS contain the first K ids $\in GS$;
Return WMQKS.
End.

Fig. 3: Proc: Algorithm WMQKNN

a binary array B_i for each data point X_i, i=1, 2, ... n: $B_i = [b_{i1}, b_{i2}, ..., b_{id}]$ in which b_{il} = 1, if $i \in KS_l$; b_{il} = 0, if $i \notin KS_l$. Once we obtain B_i, we calculate the Weighted-multi-query-distance X_i to Q as $WMSD(X_i, Q) = W_i * \frac{\sum_{l=1}^{d} \delta_{il} * b_{il}}{(\sum_{l=1}^{d} b_{il})^2}$, where δ_{il} is the difference between X_i and the average position ($\overline{q_l} = \frac{\sum_{j=1}^{m}(WQ_j * q_{jl})}{\sum_{j=1}^{m}(WQ_j)}$) of Q on D_l, b_{il} is either 1 ($i \in KS_l$) or 0 ($i \notin KS_l$).

From the definition of WMSD(X_i, Q) we can see that only those dimensions on which X_i is close enough to Q are chosen for calculation. We present the WMQKNN algorithm in figure 3.

In this approach, we keep track of the information of all data points and all query points which occupies $O(n + m)$ space. We sort the differences between data points and Q_j on each dimension. The time required is $O(dnmlogn)$.

V. Experiments

In this section we present the experimental results on both synthetic and real data sets, which are run on Intel(R) Pentium(R) 4 with CPU of 3.39GHz and Ram of 0.99 GB.

A. Experiments on synthetic data sets

We design a synthetic data generator to produce data sets with different size and dimensionality. The sizes of the data sets vary from 20,000, 30,000... to 80,000, with the gap of 10,000 between each two adjacent data set sizes. The dimensions of the data sets vary from 10, ... to 80, with the gap of 10 between each two adjacent numbers of dimensions. The value of query set size m varies from 1,2,...,10. The value of K varies from 3,4,...,10.

We conducted various experiments on synthetic data sets. Here we present experiments to demonstrate the performance difference using different way of calculating the weight W_i for each X_i, i=1,2,...,n.

Figure 4 shows the change of algorithm accuracy when the data size increases from 20000 to 80000 using the W_i defined in formula (2). We can see that when there are more data points, more irrelevant information is involved in the calculation.

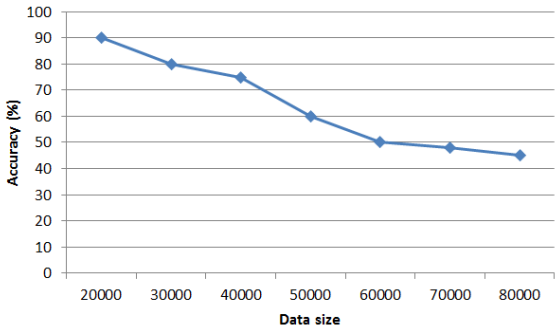

Fig. 4: The change of accuracy when data size increases using formula (2)

Figure 5 shows the change of algorithm accuracy when the data size increases from 20000 to 80000 using the W_i defined in formula (3). The performance is better than the one in figure 4, because we use maximum instead of average. However, we lose a lot of information of most dimensions.

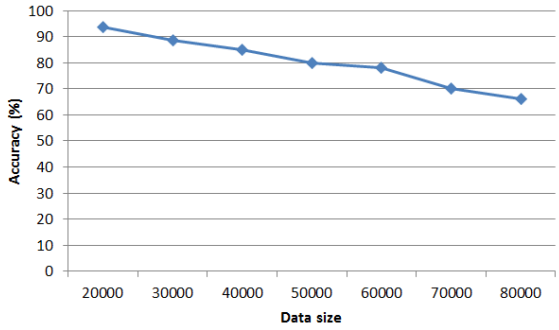

Fig. 5: The change of accuracy when data size increases using formula (3)

Figure 6 shows the change of algorithm accuracy when the data size increases from 20000 to 80000 using the W_i defined in formula (4). The performance is the best compared the previous two, because we only consider the dimensions where the data points have a lot of neighbors.

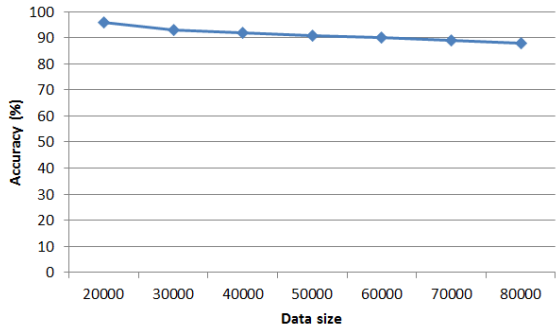

Fig. 6: The change of accuracy when data size increases using formula (4)

 ISBN: 978-1-61208-247-9

B. Experiments on real data set

We next present the experimental results of WMQKNN on real data sets.The real data sets were obtained from UCI Machine Learning Repository [1]. We compare the testing result of these data sets with other algorithms such as IGrid[5] and Frequent K-n-match algorithm [15].

The first data set is Yeast data set. It contains 1484 instances in a 8-dimensional data space. There are 10 clusters in the data set. The second data set is Wine Recognition data set. It contains the results of a chemical analysis of wines grown in the same region in Italy but derived from three different cultivars. It contains 178 instances. Each instance has 13 features which means the data set is defined in a 13 dimensional data space. Three clusters are defined with the sizes of 59, 71 and 48. The third data set is Ecoli data set for Protein Localization Sites. There are 336 instances, each of which having 7 features. 8 clusters are contained in the data set.

To generate query points, for each real data set, we randomly select data points as the candidates, and perform our algorithm using K as 6. Query point sets of various sizes are randomly selected, and for each query point, 15 data points are retrieved as the nearest neighbors. If a retrieved data point has the same class with the query point it is associated with, we call it a successful retrieval. Otherwise, we call the data point an unsuccessful retrieval. We calculate how many successful retrievals we have among the results from performing WMQKNN on these query points, and evaluate the accuracy rate. The average accuracy rate of WMQKNN algorithm is 92.6%, which is higher than the accuracy rate of IGrid (87.9%), and that of Freq. K-n-match algorithm, which is 90.8%.

VI. Conclusion and discussion

In this paper, we present our continuous work on finding nearest neighbors for multiple queries. We first analyze the data distribution on each dimension, and calculate the weight of each data point. We discussed various solutions of calculating the total weight of a data point, analyze the disadvantages of two solutions, and choose the one that calculates the average of the weight on selected dimensions. We then apply the weight to calculate the distance between a data point and the query point sets step by step.

We conduct experiments to test our approach on different data sets. We first generate synthetic data sets and demonstrate how the algorithm accuracy changes with the data size using different solutions. We then test our approach on real data sets and compare the experimental results with existing algorithms.

For the future work, we will improve our algorithm by amplifying the effect of data point weight as well as dimension weight. We will modify the way to calculate the distance between a data point and a query point set to improve the performance of our approach.

References

[1] *UCI Machine Learning Archive*. University of California, Irvine, Department of Information and Computer Science. http://kdd.ics.uci.edu. (Retrived: 01/13/2013).

[2] E. Achtert, C. Böhm, P. Kröger, P. Kunath, A. Pryakhin, and M. Renz. Efficient reverse k-nearest neighbor search in arbitrary metric spaces. In *SIGMOD '06*, pages 515–526, New York, NY, USA, 2006. ACM.

[3] C. C. Aggarwal. Towards meaningful high-dimensional nearest neighbor search by human-computer interaction. In *Proceedings of the 18th International Conference on Data Engineering, 26 February - 1 March 2002, San Jose, CA*, pages 593–604. IEEE Computer Society, 2002.

[4] C. C. Aggarwal, A. Hinneburg, and D. A. Keim. On the surprising behavior of distance metrics in high dimensional spaces. In *Proceedings of the 8th International Conference on Database Theory*, ICDT '01, pages 420–434, London, UK, UK, 2001. Springer-Verlag.

[5] C. C. Aggarwal and P. S. Yu. The IGrid index: reversing the dimensionality curse for similarity indexing in high dimensional space. In *Knowledge Discovery and Data Mining*, pages 119–129, 2000.

[6] S. Berchtold, D. A. Keim, and H.-P. Kriegel. The X-tree : An index structure for high-dimensional data. In *VLDB'96*, pages 28–39, Bombay, India, 1996.

[7] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is "nearest neighbor" meaningful? In *International Conference on Database Theory 99*, pages 217–235, Jerusalem, Israel, 1999.

[8] B. Cui, H. T. Shen, J. Shen, and K.-L. Tan. Exploring bit-difference for approximate knn search in high-dimensional databases. In *Proceedings of the 16th Australasian database conference - Volume 39*, ADC '05, pages 165–174, Darlinghurst, Australia, Australia, 2005. Australian Computer Society, Inc.

[9] D. A. White and R. Jain. Similarity Indexing with the SS-tree. In *Proceedings of the 12th Intl. Conf. on Data Engineering*, pages 516–523, New Orleans, Louisiana, February 1996.

[10] R. Fagin, R. Kumar, and D. Sivakumar. Efficient similarity search and classification via rank aggregation. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 301–312, New York, NY, USA, 2003. ACM.

[11] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *The VLDB Journal*, pages 518–529, 1999.

[12] A. Hinneburg, C. C. Aggarwal, and D. A. Keim. What is the nearest neighbor in high dimensional spaces? In *The VLDB Journal*, pages 506–515, 2000.

[13] T. Seidl and K. H.-P. Optimal multi-step k-nearest neighbor search. In *Proc. ACM SIGMOD Int. Conf. on Management of Data (SIGMOD 1998), Seattle, Washington*, pages 154–165, New York,NY,USA, 1998. ACM.

[14] Y. Shi and B. Graham. A similarity search approach to solving the multi-query problems. In *Proceedings of the 2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, ICIS '12, pages 237–242, Washington, DC, USA, 2012. IEEE Computer Society.

[15] A. K. H. Tung, R. Zhang, N. Koudas, and B. C. Ooi. Similarity search: a matching based approach. In *VLDB '06: Proceedings of the 32nd international conference on Very large data bases*, pages 631–642. VLDB Endowment, 2006.

[16] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proc. 24th Int. Conf. Very Large Data Bases, VLDB*, pages 194–205, 24–27 1998.

 ISBN: 978-1-61208-247-9