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 $q_p_1$, the solid four-point star $q_p_2$, and the solid ellipse $q_p_3$ represent different query points. Among the data points represented by the hollow dots, the data point $d_p_1$ is far from other data points, and the data point $d_p_2$ 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 $q_p_1$, $q_p_2$ and $q_p_3$ are all much closer to $d_p_2$ than to $d_p_1$. For applications of this kind, a data point closer to other data points (such as $d_p_2$) has a higher chance to be one of the K nearest neighbors of multiple query points, compared to $d_p_1$. 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
query points will be in the same data space as data points in DS are: \( Q_j = [q_{j1}, q_{j2}, \ldots, q_{jd}] \). Suppose the set of the query points is \( Q = [Q_1, Q_2, \ldots, Q_m] \), with the size of \( Q \) being \( m \). Both DS and \( Q \) are normalized. Each query point \( Q_j \) has a weight \( W_{Q_j} \) 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, \( i=1,2,\ldots,n \). 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 \( D_i \), \( i=1,2,\ldots,d \), we calculate

\[
W_{il} = |\{1 \leq j \leq n | x_{il} - x_{jl} | < \alpha, i \neq j\}|
\]

(1)

where \( \alpha \) 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 \( D_i \), \( W_{il} \) 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:

\[
W_i = \frac{\sum_{l=1}^{d} W_{il}}{d}
\]

(2)

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

\[
W_i = \max_{l=1}^{d} W_{il}
\]

(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

\[
W_i = \frac{\sum_{p=1}^{h} W_{ip}}{h}
\]

(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

\[
W = \{W_1, W_2, \ldots, W_d\}
\]

(5)

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

In the next step, we define \( \Delta_{ij} = [\delta_{i1j}, \delta_{i2j}, \ldots, \delta_{ijd}] \) as the array of differences between \( X_i \) (\( i=1,2,\ldots,n \)) and \( Q_j \) (\( j=1,2,\ldots,m \)) on all the dimensions \( (D_1, D_2, \ldots, D_d) \). \( \delta_{ijl} \) is calculated as:

\[
\delta_{ijl} = WQ_j \ast |x_{il} - q_{jl}|.
\]

(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 \( \delta_{ijl} \) based on equation (6). We next sort the data points based on \( \delta_{ijl} \) on each dimension \( D_l \), \( l=1,2,\ldots,d \), for each query point \( Q_j \), \( j=1,2,\ldots,m \). We then define \( KS_{ilj} \) as the set which contains the \( ids \) of the first \( K \) data points in the sorted list. Let \( KS_{ilj} = KS_{il1} \cup KS_{il2} \cup \ldots \cup KS_{ilm} \). We define \( KS_i \) as the set which contains the \( ids \) of \( K \) data points which appear most frequently in \( KS_{ilj} \).

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
Here we present experiments to demonstrate the performance of $K$ varies from 3, 4, ..., 10. gap of 10 between each two adjacent numbers of dimensions. The sizes of the data sets vary from 20,000, 30,000... to 80,000, 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, if $i \notin K_{Si}$ or $b_{id} = 0$, if $i \notin K_{Si}$. The value of $K$ varies from 3, 4, ..., 10, if $i \notin K_{Si}$ or $b_{id} = 0$, if $i \notin K_{Si}$.

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.

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.

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.

### Algorithm WMQKNN

**Input:** $DS$: data set, $Q$: query point set, $D$: dimensions, $K$: number of data points required, $\omega$: threshold for calculation of query point closeness

**Begin**

- For each $X_i \in DS$ and each query point $Q_j \in Q$, calculate $\Delta_{ij} = [\delta_{i1}, \delta_{i2}, ..., \delta_{id}]$ in which $\delta_{ij} = |x_{ij} - q_{ij}|$;
- Sort the data points in $DS$ based on $\delta_{ij}$ for each query point $Q_j$ and each dimension $D_i$;
- Generate $K_{Si}$ which contains the first $K$ ids in the sorted list;
- Generate $K_{Si}^l$ as union of $K_{Si}$, $i=1, 2, ..., m$;
- Generate $GS$ which contains $K$ ids from $K_{Si}^l$ with the highest frequencies;
- Calculate $W$ 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 $i \in K_{Si}$; $b_{il} = 0$, if $i \notin K_{Si}$;
- Generate $GS$ as union of $K_{Si}$, $i=1, 2, ..., d$;
- 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 $b_{id} = 1$, if $i \in K_{Si}$; $b_{id} = 0$, if $i \notin K_{Si}$. Once we obtain $B_i$, we calculate the Weighted-multi-query-distance $X_i$ to $Q$ as $WMSD(X_i, Q) = W_i + \sum_{j=1}^{m} \frac{b_{ij} \delta_{ij}}{\text{number of } X_i}$, where $\delta_{ij}$ is the difference between $X_i$ and the average position $(\frac{\sum_{j=1}^{m} \delta_{ij}}{\text{number of } X_i})$ of $Q$ on $D_i$, $b_{id}$ is either 1 ($i \in K_{Si}$) or 0 ($i \notin K_{Si}$).

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 $O(n + mn)$ 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 number of dimensions varies 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 $X_i$, $i=1, 2, ..., n$.
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


