SURVEY ON VARIOUS-WIDTH CLUSTERING FOR EFFICIENT k-NEAREST NEIGHBOR SEARCH

Sneha N. Aware[1], Prof. Dr. Varsha H. Patil[2]

[1] *PG Student, Department of Computer Engineering,Matoshri College of Engineering & Research Centre,Nashik ,Maharashtra, India.*

[2] *Head of Department of Computer Engineering, Matoshri College of Engineering & Research Centre, Nashik, Maharashtra, India.*

ABSTRACT

Over recent decades, database sizes have grown large. Due to the large sizes it create new challenges, because many machine learning algorithms are not able to process such a large volume of information. The k-nearest neighbor (k-NN) is widely used in many machines learning problem. k-NN approach incurs a large computational cost. In this research a k-NN approach based on various-width clustering is presented. The k-NN search technique is based on VWC is used to efficiently find k-NNs for a query object from a given data set and create clusters with various widths. This reduces clustering time in addition to balancing the number of produced clusters and their respective sizes.

Keyword: - *clustering,k-Nearest Neighbor, various-width clustering, high dimensional, etc.*

1. INTRODUCTION:-

The k-nearest neighbor approach (k-NN) has been extensively used as a powerful non-parametric technique in many scientific and engineering applications. To find the closest result a k-NN technique is widely used in much application area. This approach incurs a large computational area. General k-NN problem states that, suppose we have n number of objects and a Q query object, then distance from the query object to all training object need to be calculated in order to find k closest objects to Q. For example, in an image database, a user might be interested in finding the images most similar to a given query images. To find the k-NN by computing the distance between q and every object in O the techniques can be loosely classified into two categories: tree-based indexes and flat indexes.

The tree-based indexes use a binary partitioning technique to build a tree of the data sets.Each leaf node of the tree contains the objects that are close to each other. Some tree-based indexes are kd-tree [3], M-tree [7], vantage-point tree (vp-tree) [8], Cover tree [6] and Ball-tree [1]. Each tree-based index has its own technique for partitioning a data set in a recursive fashion to build a tree. The major drawback of the tree-based indexes is that a binary partition technique that may not be the appropriate way for the object distributions when the dimensionality is high. Due to large dimensional data set the internal nodes will have a high overlap with each Other[9].

Flat-indexes use the clustering techniques to directly partition the data set into a number of clusters to obtain better partitioning results, instead of creating a tree-like structure, and use the clusters to efficiently compute k-NNs. Fixed-width clustering (FWC) [10], [11] and k-means [9] are two such approaches where the feature space is directly partitioned into a number of clusters. In fixed-width clustering (FWC), it partitioning a data set into a number of clusters with fixed width. The efficiency of this algorithm is decreased if some of the produced clusters are relatively large. Due to the large cluster it increases the computational time of FWC.

In this paper we are doing survey based on the clustering methods known as various-width clustering (VWC).Various-width clustering (VWC) is used to clustering the object in data set into number of various size clusters. This result in compact and well-separated cluster in increased the effectiveness [4].

2. LITERATURE SURVEY:-

In k-Nearest neighbor approach, requires scanning whole data set and find k-NN by calculating the distances from a query object to all object in set n, that result into high computational cost. With this result, research has focused on pre-processing the data set to find k-NN by accessing only the part of dataset. The technique classified into two categories: tree-based and flat indexes.

The well-known tree-based indexes are kd-tree[5], Ball-tree[1], M-tree[7], Cover-tree[6], vp-tree[8] are explain below : Friedman et al. [12] introduced the kd-tree where data set is split into two parts in balanced binary tree, to deal with high dimensional data.

Fukunaga and Narendra [1] proposed a metric tree algorithm called the Ball tree. Ball tree compute centroid of data set, and used this centroid to partitioned the data set into two subset. The number of objects assigned to either node cannot be constrained which may result in a highly unbalanced tree structure [1]. Leaf nodes of any M-tree [7] store all data points, whereas the internal nodes store the called as routing objects. All the objects that are within a given distance from the routing object are stored in the sub-tree of the routing object called covering tree.

Beygelzimer et al. [6] proposed cover tree which is a hierarchy of levels where, the top level contain the root point and the bottom level contain every point in the metric space. Cover time output in logarithmic time to find nearest neighbor.

In vp-tree due to the partitioning based on distances from the vantage points, the objects in the leaf nodes overlap with each other nodes especially in a high dimensional data [8].
Tree-based indexes losses its effectiveness if overlap between the nodes is high and result into unbalanced tree structure. Several flat-based indexes are used e.g., k-means and fixed width clustering. With k-means, assigns sparse object to the closest cluster such that clusters are overlapping with lots of another cluster [9].
In fixed width clustering, result in poor clustering due to large number of object in some clusters require large amount of computational time for searching[10],[11].
Almalawi et al. [4] proposed a flat index that uses different widths for different clusters and width of each cluster is learned during the construction, this approach is known as various-width clustering. In this technique, dataset is partitioned into a number of clusters whose sizes are constrained by user-defined threshold. Three operations performed to generate the clusters are cluster-width learning, partitioning and merging such that they loop sequentially and executed until criteria are met. Further these operations, they performed k-NN search to obtain result.
As computing cluster for each dataset and find the nearest distance for each query object to centroid of cluster, result into high computational cost and pre-processing cost. In existing approach, k-NN search for searching distance of query object with centroid of each clusters to obtain the nearest distance required large amount of time.

3. BLOCK DIAGRAM:-

Fig1. shows the system architecture for k-NN search. As input require to the system is high dimensional data set, which is further processing in the data normalization, for example, we have data in column with height and weight then we have to normalized the data in single unit for comparison.
The next step is to produce the number of cluster by using various-width algorithm which perform three operations are data partitioning, data fixed width clustering and data merging. In partitioning process, partition a data set into number of clusters using large width to resolve the issue of clustering in high dimensional data space. In data fixed width clustering, compute radius of clusters and its nearest distance from query object. In merging process, after partitioning the cluster it check the minimum distance between centroid of two clusters and their respective radius. If distance between centroid of cluster with radius of cluster is less than radius of another cluster then merge the cluster with other cluster.
The next step is k-NN search, for query object so that we can easily map (i.e. compare) with every cluster object. The comparison task performed parallel executes which result fast comparison. k-NN process executed using GPU, allocate memory on CUDA, then transfer the data on it. Decide how many threads and blocks have to launch to run the program. Using CUDA which is parallel computing platform and programming model that increase the computing performance using the graphics processing unit (GPU). Using GPU reduces the searching time, and do not want to compare with every cluster object, because with parallel k-NN we parallel launch the thread and solve problem parallel which result in reduce the comparison task.

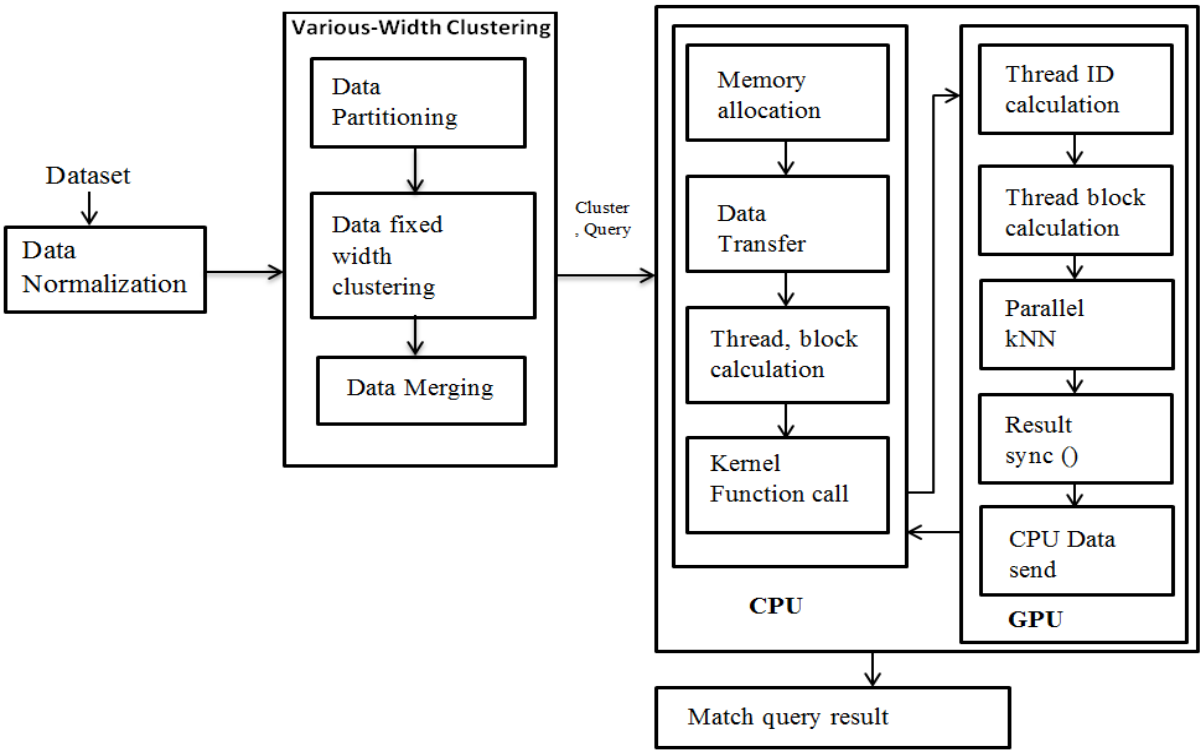

Fig-1: Block diagram for k-NN Search based on VWC

4. SYSTEM ANALYSIS:-

Various-width clustering for k-nearest neighbor(k-NN) search consist of three operations need to be perform to find the exact k-nearest neighbor object to the query object from high dimensional data set. The three operations are fixed-width clustering (FWC), various-width clustering (VWC) and k-NN search for query object.

4.1 Fixed-width clustering (FWC):

The Fixed-width clustering (FWC) is for partitioning a data set into a number of clusters with fixed width radius r. Steps for fixed-width clustering are as follows:
1. Input: List of objects, pre-defined radius of cluster
2. Initialized: set of clusters, their centroid and width to null and number of created cluster to be zero(n=0)
3. for first object j_i in U.objects do
4. if number of created cluster are zero(n=0) then
5. create first cluster(n+=1)
6. putj_i in C_n; put j_i in Centroid
7. else
8. Find the ID and distance of the closest cluster to assign j_i
9. Check condition C_i< w and n > 0 if ok then continue
10. else create new cluster C_{i+1} and set j_i as its centroid
11. Output: fixed-width clusters.

4.2 Various-width clustering (VWC):

Various-width clustering is used to partitioning the data set into a number of clusters whose sizes are varied in sized and constrained by user-defined threshold. As shown in Fig.1 the various-width clustering consist of three processes are: Data partitioning, Data fixed width clustering and Data merging.
Following algorithm summarizes the steps required to perform various-width clustering are as follows:
Input required to this algorithm is data set, and user-defined threshold value. Initially, whole data set is consider as a cluster and its centroid and width set with zero as shown in line 4. β is denoted as threshold such that any cluster that exceed the value of cluster will be further partitioned. In line 7, find out the cluster size. In Partitioning procedure, the function *LargestCluster* return the largest cluster U from clusters shown on line 14. If size of U is greater than β then width *ω* is calculated for partitioning U. If width is zero then assigned U as non-partition mention in lines 18-23. Otherwise, Algorithm for FWC is called to partition U(line 21). If a produced cluster is only one it means that value of *ω* is very large and it should be minimized as in line 27. Otherwise, new cluster is produced from U are added to clusters, and largest cluster again pull from clusters as shown in line 22-25. Partitioning the largest cluster until cluster is less than β.
In partitioning process contain large cluster that create cluster that is totally contained in another cluster. Therefore, merging process is used to decrease the number of produced cluster and increased the performance of k-NN search. In merging process, the list of all clusters is iterated to obtain the IDs of parents and child clusters (lines 14-40). All child clusters are associated with its parent hence this child cluster is removed from list clusters. Not removed the child processes that are parent of other clusters and merging is used.

Algorithm: Various-width clustering (VWC)

```
1  Input: Data
2 Input: β
3 Output: Clusters
4 Clusters ←Φ; add(Clusters,[Data, zeros; 0]);
5 finished← 0;
6 whilefinished == 0do
7 ClsSize←Clusters.getSize/*The number of clusters */
8 Partitioning(Clusters,β);
9 Merging(Clusters,β);
10 if| LargestCluster(Clusters)|≤ β or
Clusters.getSize == ClsSize then
11 finished← 1
12 return [Clusters];
13 ProcedurePartitioning(Clusters,β)
14 U ←LargestCluster(Clusters);
15while| U.objects |>βdo
16  ω ← using eq. (1);
17if(ω== 0)then
18        U.nonPartitioned(1);
19        update(Clusters,U);
20        continue;
21 <tmpClusters> ←Algorithm 1(U,w);
22ifClusterNum(tmpClusters)> 1 then
```

```
23        remove(Clusters, U);
24        add(Clusters,tmpClusters);
25        U ←LargestCluster(Clusters);
26else
27 ω ← ω − (ω ∗ 0.1);
28go to line 21
29 ProcedureMerging(Clusters,β)
30 MergingList← Φ/* list of tuples < */
                  /* childClusterID, parentClusterID> */
31foreachU in Clustersdo
32 j ← using eq. (2) and eq. (3);
          /* ID of cluster contained U */
33        if j ≠ 0 then
34        put <U.getID, j > in MergingList;
35whileMergingList≠ ∅ do
36        foreachtuple in MergingListdo
37        <i,j>← tuple;
38        if!isParent(MergingList, i)then
39        MergeClus(Clusters, i, j);
40         remove tuple from MergingList;
```

4.3 The k-NN search:

After finding the number of cluster with query object have to find the top k-nearest object from N using k-NN search.

Algorithm for k-NN search is as follow:

```
1Input: clusters
2 Input: k
3 Input: q  /* The query object */
4Output: N
5 cluID ← Φ;
/* list of tuples <clusterID, distance > sorted
by distance in ascending order */
6 cluID ←ClusAscOrder(clusters; q);
7 tmpU← Φ;
8 foreach{clusterID, distance} in cluIDdo
9 U ←get(clusters; clusterID);
10 tmpU ←tmpU∪ U.objects
11remove <clusterID, distance> from cluID;
12if| tmpU |> kthen
13          break;
14 T ← NN_k(p, tmpU, k);
          /* list of tuples <objectID, distance > */
15 N ← Φ; Z ← Φ;
16 foreach{clusterID, cluDis} in cluIDdo
17 U←  get(clusters, clusterID);
18 foreach{objID, objDis} in T do
19        if(cluDis− U.radius)<objDis then
20        put clusterID in Z;
21 break;
22foreachclusterID in Z do
23U←  get(clusters; clusterID);
24 put NN_k(p, U.objects, k) in N;
25 N= N ∪ T;
26 return top k-nearest objects from N;
```

Algorithm summarises the k-NN search for query object throughout the clusters. As shown in the algorithm inputs are cluster, k be the number of nearest neighbours that need to be found and query object q.

When query object is received the function ClusAscOrder returns the sorted list of clusters ID and their distances to q, the list is named as cluID. A tmpU be the temporal cluster object to represent the closest cluster to q. For each cluster a separate thread is launched and find out the closest clusters. If the size of tmpU is less than k, the object of second closest cluster are merged in tmpU, so that it reduce the comparison time with each cluster. The process continue until size of tmpU become greater than k as mention at Line 12. The final step is to return the top k objects from N as the "exact" k-NNs for q (Lines 22-26).

5. CONCLUSIONS

This work presented a k-nearest neighbor searchingapproach, using various-width clustering algorithm.By using various-width clustering it shows us how to produce the number of cluster and with k-NN search algorithm give effective searching technique to find k nearestneighbors. This approach is able to produce compact and well-seaparatedclusters from high dimensional data of various distributions.VWC algorithm shares efficiency feature in producingless overlapping clusters with non-binary partitioning techniques compare to FWC,

6. ACKNOWLEDGEMENT

It gives me an immense pleasure to express my sincere and heartiest gratitude towards our guide Prof. Dr.Varsha H. Patil, Department of ComputerEngineering for her guidance, encouragement and moral support during the course of our work. She has proven to be an excellent mentor and teacher. This work is also the outcome of the blessings, guidance and support of our parents, family members and friends. Lastly our thanks to all who have contributed indirectly in completion of this work

7. REFERENCES

1] K. Fukunaga and P. M. Narendra, A branch and bound algorithm for computing k-nearest neighbors, IEEE Trans. Comput., vol. C-100, no. 7, pp. 750-753, Jul. 1975.

2]S. A. Nene and S. K. Nayar, "A simple algorithm for nearest neighbor search in high dimensions," IEEE Trans. Pattern Anal. Mach. Intell., vol. 19, no. 9, pp. 989–1003, Sep. 1997.

3] R. F. Sproull, "Refinements to nearest-neighbor searching in k-dimensional trees," Algorithmica, vol. 6, no. 1–6, pp. 579–589, 1991.

4] Abdul Mohsen Almalawi, AdilFahad, ZahirTari, Muhammad AamirCheema, and Ibrahim Khalil, "kNNVWC: An Efficient k-Nearest Neighbors Approach Based on Various-Widths Clustering," IEEE Trans.Knowl. Data Eng., vol. 28, no.1, pp. 68-81, Jan. 2016.

5] J. H. Friedman, F. Baskett, and L. J. Shustek, "An algorithm for finding nearest neighbors," IEEE Trans. Comput., vol. C-24, no. 10, pp. 1000–1006, Oct. 1975.

6] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," in Proc. 23rd Int. Conf. Mach. Learning, 2006, pp. 97–104.

7] P. Patella, M. Ciaccia, and P. Zezula, "M-tree: An efficient access method for similarity search in metric spaces," in Proc. 23rd Int. Conf. Very Large Data Bases, 1997, pp. 426–435.

8] P. N. Yianilos, "Data structures and algorithms for nearest neighbor search in general metric spaces," in Proc. 4th Annu. ACMSIAM Sym. Discr. Algorithms, 1993, pp. 311–321.

9] W. Xueyi, "A fast exact k-nearest neighbors algorithm for high dimensional search using k-means clustering and triangle inequality," in Proc. Int. Joint Conf. Neural Netw., 2011, pp. 1293–1299.

10] E. Eskin, A. Arnold, M. Prerau, L. Portnoy, and S. Stolfo, "A geometric framework for unsupervised anomaly detection," in Applications of Data Mining in Computer Security. New York, NY, USA: Springer, 2002, pp. 77–101.

11] M. J. Prerau and E. Eskin, "Unsupervised anomaly detection using an optimized k-nearest neighbors algorithm," Undergraduate thesis, Columbia Univ., New York, NY, USA, Dec. 2000.

12] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," ACM Trans. Math. Softw., vol. 3, no. 3, pp. 209–226, 1977.