# A Novel Indexing based Decision tree model on high Dimension Data Stream

G.Geetha

*M.tech Student in CSE Department, Swarnandhra College of Engineering & Technology,Narsapuram,Andhra Pradesh,India.*

Dr. M Nagabhushana Rao

*Professor in CSE Department, Swarnandhra College of Engineering & Technology, Narsapuram, Andhra Pradesh, India.*

***Abstract:***

*As the dimension increases, the complexity to find the nearest neighbor also increases. In one dimension we can find the nearest point very efficiently. But when the dimension increases, the search efficiency depends on how many points we have to search. In this thesis, we propose a simple and efficient technique on how to organize the points in the high dimensional space which also allows searching the nearest point efficiently. We have used a memory efficient database organization of the points in the high dimension, and the searching algorithm is based on limiting the search within some particular indexes of the database. A comparative analysis reveals that our database organization is faster than Etree, R-tree , and the searching algorithm is also very efficient.*

***Keywords*** *– Watermark, Security, Encode and Decode.*

## I. INTRODUCTION

The nearest neighbor search is an important field of research which has been widely used in multimedia database, pattern recognition, computational engineering etc. The nearest neighbor problem can be formulated as: given a query point q, find the point t that has the shortest distance from q. In order to search the nearest point, we must store the points in the database [2] and then search. So it requires efficient database management to search the nearest point. It is also required what kind of information of the points (i.e. coordinate etc) we have to store in the database. There are several high dimensional database organizations such as SR-tree [5], R-tree [23], K-d tree [25] etc. Besides these, the authors [1] used an ellipsoid cluster to store and search the nearest neighbor in the high dimension and d (d = number of dimensions) number of one-dimensional arrays [2] to perform storage and searching. In our approach, we have considered a very simple organization of the database in the high dimensional space. We have proposed only one database which will contain the high dimensional values of the points and also their distance from the origin. We have used an efficient searching algorithm to find the nearest
neighbor or point by using the information of the database in a particular region surrounding the query point. Traditional methods consider a search in a one-dimensional array. In the worst case, we have to search all the values in the array to find the smallest value stored in the array. So its complexity will be O(n) in the worst case. If we search in this way in higher dimensional space, it will be an impractical solution. We need an efficient algorithm to search for the nearest point for a query point in the high dimensional space. In this thesis, we have developed a technique that searches for the nearest point in the high dimensional space in an efficient way.

## II. LITERATURE SURVEY

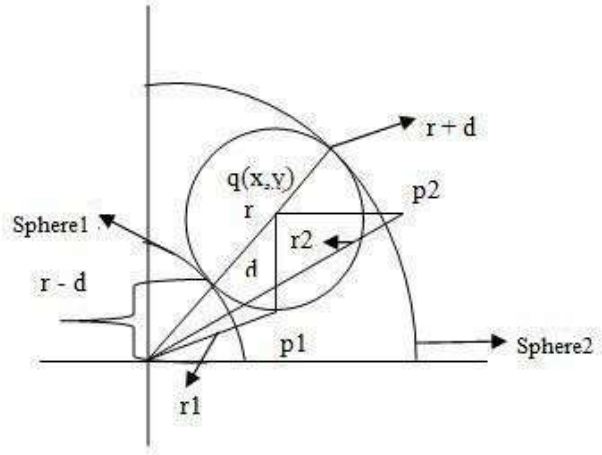
The authors have proposed another type of storage for the points in the high dimension [2]. At first, they have stored the points in d number (d = number of dimensions) of one-dimensional arrays where the jth array contains the jth coordinate of the point in d dimensional space. Here the searching algorithm sorts these one- dimensional arrays at first. Then it creates four candidate lists from these arrays which contain the query point. After this, it exhaustively searches for the nearest neighbor inside a cube around the query point.

Besides this database storage, there are some tree based data structures which have been widely used in high dimensional database indexing. These are SR-tree [5], R-tree [23], K-d tree [35] etc. R-tree is a tree data structure used to index multidimensional

information using the minimum bounding rectangle (MBR) of the data [23]. The minimum bounding rectangle means the maximum extension of a two-dimensional object

point, line, polygon) in its two coordinate (x,y) system. The entries for each node in the R-tree are variable. The entries for each node have pointers to the child node and the MBR of all entries with in this child node. The insertion and deletion ensures that the nearby elements reside inside the same leaf node by using the MBR from the nodes. A new element will be inserted in a leaf node which requires the least extension of its MBR. In the same way, the searching algorithm will prune away unnecessary nodes by examining the MBR of the element [28]. The SR-tree is also similar to R-tree where it uses the intersection of the bounding sphere and the bounding rectangle of the data set [5]. The K-d tree is a binary tree in which each node is a K dimensional point and associated with one of the k dimensions with the hyper plane perpendicular to that dimension's axis [24]. Every non leaf node in K-d tree divides the space into two sub trees where the left sub trees have values less than the right sub trees for the dimension with which that node is associated with. In our approach, we have considered a very simple approach of organizing the points in one single database and searching through the database efficiently to find the nearest neighbor.

In the database, the index of the points represents the distance of the points from the origin. It is a must to get the index position of the query point at first. The index position of the query point can be easily determined by comparing the distance of the query point from the origin with that of the points in the database. At this point, the index position of the query point is known (Lets it is i). Now the nearest point to the query point lies within certain range from the index position of the query point to the nearby points. From the index position of the query point, the search algorithm can move upward (i.e. index position i-1, i-2, … 0) and compare the distance between each of the points and the query point to find the nearest point in that direction. It can also go downward (i.e. index position i+1, i+2, … i+N) and compare the distance between each of the points and the query point to find the nearest point in the downward direction. Now it can compare these two distances from the upward and downward direction of the database and get the nearest point for the database. The upward direction means the points which are closer to the origin and the downward direction means the points which are farther away from the origin. In each of this direction, there can be millions of points to compare. We have proposed a technique which will not compare the distance of each of the points in the database except some points in a certain range in the database. The authors [2] represent a searching technique which searches the nearest point around the query point in a certain limited region in the high dimensional space. They have used a probabilistic parameter ϵ to calculate the surrounding region of the query point. We have proposed a deterministic approach which will not compare the distance of each of the points in the database except some points in a certain range in the database.



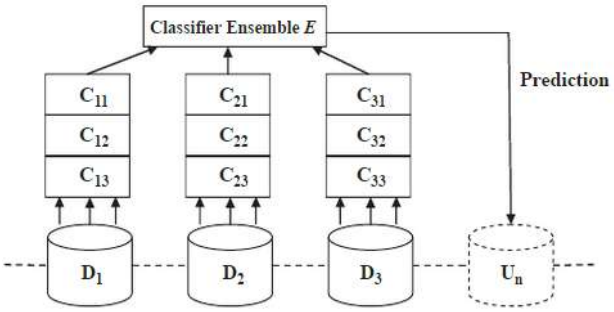

In order to explain clearly, let's consider a two dimensional space. Suppose the query point belongs to the index i of the database (q(x,y) in fig 1). We can have two points from the database at the index i-1 and i+1 (p1 and p2 respectively in fig 1). We can also calculate the minimum Euclidean distance from these two points to the query point at the position i and let's it is considered as d[1].

The two spheres around the circle (in fig1) limit the search in this region. In the downward direction of the database, the algorithm will stop checking with the distance of the points which has distance > r + d from the origin as it is in figure 1. In the upward direction, it will stop at the position of the database where the distance is < r - d. This ensures that the algorithm has checked all the points inside the circle. In each of the

direction, the algorithm checks whether the points reside inside the square and then the circle. Then it compares the distance and gets the minimum one. The IR2-tree is the first access method for answering NN queries with keywords. As with many pioneering solutions, the IR2-tree also has a few drawbacks that affect its efficiency. The most serious one of all is that the number of false hits can be really large when the object of the final result is far away from the query point, or the result is simply empty. In these cases, the query algorithm would need to load the documents of many objects, incurring expensive overhead as each loading necessitates a random access.

## III. PROPOSED WORK

One point is for the upward direction and another one for the downward direction. Now inside these points we will search and compare the distance between the point and the query point if it exists in the hypercube and then the hyper sphere. If the distance is less than the existing distance, the final distance will be updated.




Variables: N, O, P, and R are nodes in the hierarchy.

I is an unclassified instance.

A is a nominal attribute.

V is a value of an attribute.

**Incorporate(N, I)**

update the probability of category N.

For each attribute A in instance I,

For each value V of A,

Update the probability of V given category N.

**Create-new-terminals(N, I)**

Create a new child M of node N.

Initialize M's probabilities to those for N.

Create a new child O of node N.

Initialize O's probabilities using I's value.

**Merge(P, R, N)**

Make O a new child of N.

Set O's probabilities to be P and R's average.

Remove P and R as children of node N.

Add P and R as children of node O.

Return O.

Split(P, N)

Remove the child P of node N

For each Fragmented Node Instances perform the FCM technique as follows:

**Improved Fuzzy C-means**

For each point $x$ we have a coefficient giving the degree of being in the $k$th cluster $u_k(x)$. Usually, the sum of those coefficients is defined to be 1, so that$u_k(x)$ denotes a probability of belonging to a certain cluster:

$$\forall x \quad \sum_{k=1}^{\text{num. clusters}} u_k(x) = 1.$$

With fuzzy $c$-means, the centroid of a cluster is the mean of all points, weighted by their degree of belonging to the cluster:

$$\text{center}_k = \frac{\sum_x u_k(x)^m x}{\sum_x u_k(x)^m}.$$

The degree of belonging is related to the inverse of the distance to the cluster

$$u_k(x) = \frac{1}{d(\text{center}_k, x)},$$

then the coefficients are normalized and fuzzyfied with a real parameter $m > 1$ so that their sum is 1.

Uses a specific distance measure called the Ward distance:

$$d(X,Y) = \frac{n_X \cdot n_Y}{n_X + n_Y} \cdot |mean_X - mean_Y|^2$$

where $n_X$ indicates the number of instances in cluster X (and likewise for Y).

In each iteration, clusters which have the shortest distance are combined.

The purpose of feature weighting for clustering is to assign proper weight values for all features according to their importance to the clustering quality.After feature weighting is applied , $disw(x_m, c_k)$ ,which means distance with feature weighting and is formulated as:

$$disw(x_m, c_k) = \sum_{n=1}^{N} w_n \times d(x_{mn}, c_{kn})$$

$$disw(x_m, c_k) = \sum_{n=1}^{N} w_n \times |x_{mn} - c_{kn}|$$

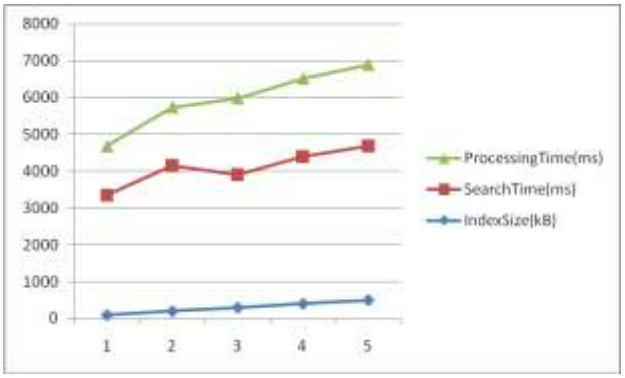
Where $w_n$ is the weight of feature $f_n$ and $w = \{w_1, w_2 ..... w_n\}$ is the set of n feature weights .

Our searching algorithm has the linear time complexity because it can search and compare the distance of the points in a single loop. If there is n1 number of points inside the hyper sphere, then the complexity will be O(n1). The searching approach sorts the points at first and then it searches for the nearest point for a query point.

## IV.Experimental Results

All experiments are performed with the configurations Intel(R) Core(TM)2 CPU 2.13GHz, 2 GB RAM, and the operating system platform is Microsoft Windows XP Professional (SP2).

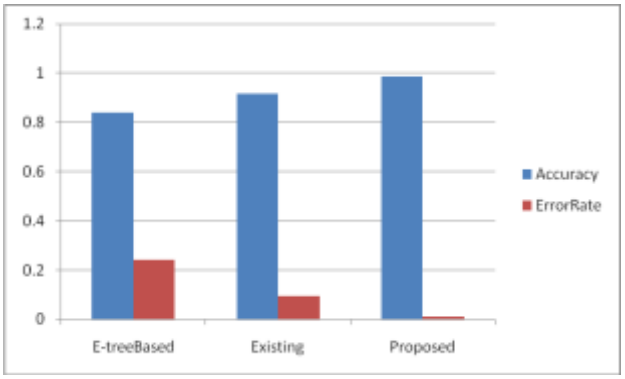
| IndexSize(kB) | SearchTime(ms) | ProcessingTime(ms) |
|---|---|---|
| 100 | 3252 | 1334 |
| 200 | 3946 | 1594 |
| 300 | 3599 | 2088 |
| 400 | 4000 | 2116 |
| 500 | 4187 | 2199 |




**Search time and process time comparison**

| Algorithm | Accuracy | ErrorRate |
|---|---|---|
| E-treeBased | 0.84 | 0.24 |
| Existing | 0.917 | 0.096 |
| Proposed | 0.986 | 0.012 |

**Proposed algorithms vs Existing models**




**Proposed algorithms vs Existing models**

### V. CONCLUSION

Within this paper we studied the best way to efficiently find within the number of strings those the same to a given string. We made two contributions. First, we developed new algorithms that could greatly improve performance existing algorithms. Second, we studied how you can integrate existing filtering techniques with one of these algorithms, and showed that they ought to be used together judiciously, ever since the avenue for doing the combination can greatly affect the performance. We have used a memory efficient database organization of the points in the high dimension, and the searching algorithm is based on limiting the search within some particular indexes of the database. A comparative analysis reveals that our database organization is faster than K-d tree, R-tree , and the searching algorithm is also very efficient.

## 7.REFERENCES

[1] A. Arasu, V. Ganti, and R. Kaushik, "Efficient Exact Set-Similarity Joins," in *VLDB*, 2006, pp. 918–929.
[2] R. Bayardo, Y. Ma, and R. Srikant, "Scaling up all-pairs similarity search," in *WWW Conference*, 2007.
[3] D. Deng, Y. Jiang, G. Li, J. Li, and C. Yu. Scalable column concept determination for web tables using large knowledge bases. *PVLDB*, 6(13):1606–1617, 2013.
[4] J. Feng, J. Wang, and G. Li. Trie-join: a trie-based method for efficient string similarity joins. *VLDB J.*, 21(4):437–461, 2012.
[5] E. H. Jacox and H. Samet. Metric space similarity joins. *ACM Trans. Database Syst.*, 33(2), 2008.
[6] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani, "Robust and Efficient Fuzzy Match for Online Data Cleaning," in *SIGMOD*, 2003, pp. 313–324.
[7] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. In *OSDI*, pages 137–150, 2004.
[8] Y. Kim and K. Shim. Parallel top-k similarity join algorithms using mapreduce. In *ICDE*, pages 510–521, 2012.
[9] F. Li, B. C. Ooi, M. T. ¨Ozsu, and S. Wu. Distributed data management using mapreduce. *ACM Comput. Surv.*, 2014.
[10] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava, "Approximate string joins in a database (almost) for free," in *VLDB*, 2001, pp. 491–500.