ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 1, January 2014

Comparison of Brute-Force and K-D Tree Algorithm

Deepika Verma[1], Namita Kakkar[2], Neha Mehan[3]

M.tech Student, Department of Computer Science & Engineering, RBIEBT (Kharar, Punjab), India[1]

Assistant Professor, Department of Computer Science & Engineering, RBIEBT (Kharar, Punjab), India[2]

Assistant Professor, Department of Computer Science & Engineering, RBIEBT (Kharar, Punjab), India[3]

Abstract: Data mining may be viewed as the extraction of the hidden predictive information from large databases, is a powerful new technology with great potential to analyze important information in the data warehouse. Nearest neighbor search (NNS), also known as proximity search, similarity search or closest point search, is an optimization problem for finding closest points in metric spaces. Brute-force search is a very general problem-solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement. A Brute-force algorithm for string matching problem has two inputs to be considered: pattern and text. A k-d tree, or k-dimensional tree, is a data structure used for organizing some number of points in a space with k dimensions. K-d trees are very useful for range and nearest neighbour searches.

In this paper, we studied and compared k-d tree algorithm and brute force algorithm on various levels. The use of the approximate k-nearest neighbour with K-d Tree data structure and comparing its performance attributes to the brute-force approach. In approximate nearest neighbour to evaluate and compare the efficiency of the data structure when applied on a particular number of points, distance and execution time. The work performed between two techniques and select the best one. The result of the work performed in this paper revealed better performance using the k-d tree, compared to the brute-force approach. The aim of the algorithm is to make faster, more accurate and efficient data structure primarily depends on a particular data set. It can be further expanded as by changing the k-d tree traversal technique. We have proposed a new modified traversal technique for k-d tree.

Keywords: Data Mining, Data Structure, Binary Search Tree, Nearest neighbour, Approximate K-NN, K-d tree, Brute-force.

I. INTRODUCTION

A focus of this research is to improve performance of the KNN approach and to demonstrate its performance in a real-world problem. Another objective of this paper is to test the performance of the existing K-d tree approach. There are a large number of methods, techniques and algorithms that organize, manage, and maintain objects in a structured manner. The purpose of k-nearest neighbour (kNN) is to calculate the nearest neighbour on the basis of value of k, that specifies how many nearest neighbours are to be considered to define the class of a sample data point [1]. The k-nearest neighbour join combines each point of one point set with its k nearest neighbours [2]. The general model of a KNN query is that the user gives many query types such a point query in multidimensional space and a distance metric for measuring distances between points in this space. The system is then tried to find the K closest or nearest answers in the database from the submitted query (i.e. query point). Generally distance metrics may include: Euclidean distance [3].

Given a set of n points in a d-dimensional space, the k-d tree is constructed recursively as follows. First, one finds a median of the values of the ith coordinates of the points (initially, i = 1). That is, a value M is computed, so that at least 50% of the points have their ith coordinate greater-or-equal to M, while at least 50% of the points have their coordinate smaller than or equal to M. The value of x is stored, and the set P is partitioned into PL and PR , where PL contains only the points with their ith coordinate smaller than or equal to M, and |PR| = |PL |±1. The process is then repeated recursively on both PL and PR, with i replaced by i + 1 (or 1, if i = d). When the set of points at a node has size 1, the recursion stops. The kNN implementation can be done using k-d tree algorithm increase the speed of basic KNN algorithm [4], [5].

The simplest version of the kNN algorithm is the 'Brute Force' implementation and consists of three stages. The first stage is to calculate all of the `distances' from each query point to every reference point in the training set. The second stage is to sort these distances and select the k objects that are the closest from which the third and final stage of classification can be performed. More formally, KNN finds the K closest (or most similar) points to a query point among N points in a d- dimensional attribute (or feature) space. K is the number of neighbours that are considered from a training data set and typically ranges from 1 to 11 [1].

A. Data Mining

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 1, January 2014

Data mining is an interdisciplinary subfield of computer science which involves computational process of large data sets patterns discovery. The goal of this advanced analysis process is to extract information from a data set and transform it into an understandable structure for further use. The methods used are at the juncture of artificial intelligence, machine learning, statistics, database systems and business intelligence. Data Mining is about solving problems by analyzing data already present in databases. Data mining is the process of analyzing data from different perspectives and summarizing it into useful information. Data mining is also known as Knowledge Discovery in Data (KDD) [6]. Data mining is a powerful tool that can help to find patterns and relationships within our data. Data mining discovers hidden information from large databases. To ensure meaningful data mining results, we must understand our data.

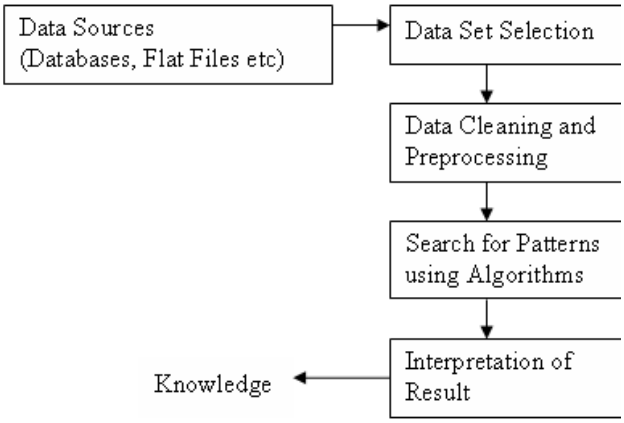


Figure: 1 shows the Process of Data Mining.

Following can be done using Data mining process [7].

a. It take out, change, and load data onto the data warehouse system.

b. It store and manage the data in a database system.

c. It provides data access to business analysts, information technology professionals and other persons.

d. Analysis of the data can be done by application software.

It represents the data in an understandable format, such as a graph or table.

B. Data Structure

Data structures are ways to organize data (information). Data structure is a particular way of storing and organizing data in a computer so that it can be used efficiently. Data structures are a method of representing of logical relationships between individual data elements related to the solution of a given problem. Data structures are the most convenient way to handle data of different types including abstract data type for a known problem. The components of data can be organized and records can be maintained. Further, the record formation leads to the development of abstract data type and database systems. In data structures, we also have to decide on the storage, retrieval and operation that should be carried out between logically related items. For example, the data must be stored in memory in computer-understandable format, i.e. 0 and 1 and the data stored must be retrieved in human-understandable format, i.e. ASCII. In order to transform data various operations have to be performed [8].

C. Binary Search Tree

A binary tree in which the nodes are labeled with elements of an ordered dynamic set and the following BST property is satisfied: all elements stored in the left subtree of any node x are less than the element stored at x and all elements stored in the right subtree of x are greater than the element at x.

Figure shows a binary search tree. Notice that this tree is obtained by inserting the values 13, 3, 4, 12, 14, 10, 5, 1, 8, 2, 7, 9, 11, 6, 18 in that order, starting from an empty tree [9].

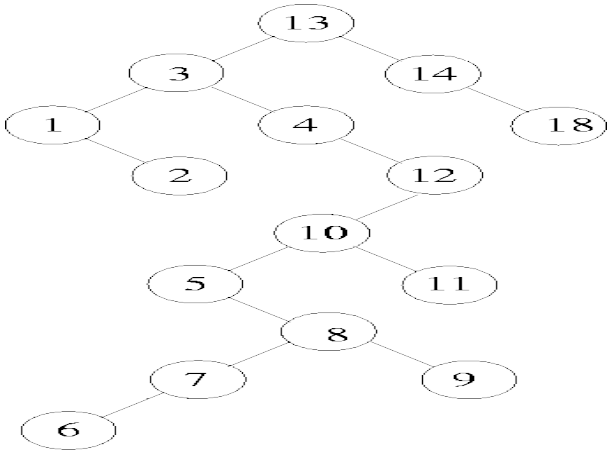


Figure: 2 An example of a binary search tree

1. Search is straightforward in a BST. Start with the root and keep moving left or right using the BST property. If the key we are seeking is present, this search procedure will lead us to the key. If the key is not present, we end up in a null link.
2. Insertion in a BST is also a straightforward operation. If we need to insert an element x, we first search for x. If x is present, there is nothing to do. If x is not present, then our search procedure ends in a null link. It is at this position of this null link that x will be included.
3. If we repeatedly insert a sorted sequence of values to form a BST, we obtain a completely skewed BST. The height of such a tree is n - 1 if the tree has n nodes. Thus, the worst case complexity of searching or inserting an element into a BST having n nodes is O(n).

Binary search tree (BST), sometimes also called an ordered or sorted binary tree, is a node-based binary tree data structure which has the following properties:

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 1, January 2014

1. The left subtree of a node contains only nodes with keys less than the node's key.
2. The right subtree of a node contains only nodes with keys greater than the node's key.
3. The left and right subtree each must also be a binary search tree.
4. There must be no duplicate nodes.

Generally, the information represented by each node is a record rather than a single data element. However, for sequencing purposes, nodes are compared according to their keys rather than any part of their associated records [10].

D. Nearest Neighbour

Nearest neighbour search (NNS) also known as proximity search, similarity search or closest point search, is an optimization problem for finding closest points in metric spaces. The problem is: given a set S of points in a metric space M and a query point q ∈ M, find the closest point in S to q. In many cases, M is taken to be d-dimensional Euclidean space and distance is measured by Euclidean distance or Manhattan distance [11]. For some applications we may have N data-points and wish to know which is the nearest neighbor for every one of those N points. This could of course be achieved by running a nearest-neighbor search once for every point, but an improved strategy would be an algorithm that exploits the information redundancy between these N queries to produce a more efficient search. Nearest neighbor is a technique that is quite similar to clustering - its essence is that in order to predict what a prediction value is in one record look for records with similar predictor values in the historical database and use the prediction value from the record that it "nearest" to the unclassified record [12].

E. K-nearest neighbors

K-nearest neighbour uses the training set directly to classify an input when an input is given. When using a k-nearest neighbor algorithm on an input with d attributes the input is classified by taking a majority vote of the k (where k is some user specified constant) closest training records across all d attributes. "Closest" as used means the distance an attribute is away from the same attribute of the training set, using some specified similarity metric. k-nearest neighbour (kNN) in which nearest neighbour is calculated on the basis of value of *k*, that specifies how many nearest neighbours are to be considered to define class of a sample data point. The training points are assigned weights according to their distances from sample data point. But still, the computational complexity and memory requirements remain the main concern always. The general model of a KNN query is that the user gives many query types such a point query in multidimensional space and a distance metric for measuring distances between points in this space [13]. The system is then tried to find the *K* closest or nearest answers in the database from the submitted query (i.e. query point). Generally distance metrics may include: Euclidean distance, Manhattan distance, etc.

K-nearest neighbour algorithm (KNN) is part of supervised learning that has been used in many applications in the field of data mining, statistical pattern recognition and many others. It is usual to use the Euclidean distance.

The algorithm on how to compute the K-nearest neighbors is as follows:

1. Determine the parameter K = number of nearest neighbors beforehand.
2. Calculate the distance between the query-instance and all the training samples. You can use any distance algorithm.
3. Sort the distances for all the training samples and determine the nearest neighbors based on the K-th minimum distance.
4. Since this is supervised learning, get all the Categories of your training data for the sorted value which fall under K.
5. Use the majority of nearest neighbors as the prediction value of the query instance [14].

6. Approximate Nearest Neighbour (ANN)

In some applications it may be acceptable to retrieve a "good guess" of the nearest neighbor. In those cases, we can use an algorithm which doesn't guarantee to return the actual nearest neighbor in every case, in return for improved speed or memory savings. Often such an algorithm will find the nearest neighbor in a majority of cases, but this depends strongly on the dataset being queried [15].

F. K-d Tree

A k-d tree, or k-dimensional tree, is a data structure used for organizing some number of points in a space with k dimensions. It is a binary search tree with other constraints imposed on it. K-d trees are very useful for range and nearest neighbour searches. The root-cell of this tree represents the entire simulation volume. The other cells represent rectangular sub-volumes that contain the mass, center-of-mass, and quadrupole moment of their enclosed regions. It was one of the early structures used for indexing in multiple dimensions. Each level of K-d tree partitions the space into two partitions, the partitioning is done along one dimension of the node at the top level of the tree, along another dimension in nodes at the next level, and so on, iterating through the dimensions. The partitioning proceeds in such a way that, at each node, approximately one half of the points stored in the subtree fall on one side, and one half fall on the other. Partitioning stops when a node has less than a given maximum number of points [16].

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 1, January 2014

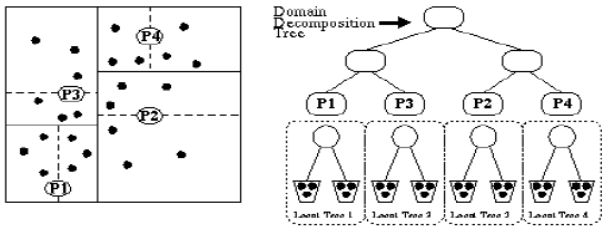


Figure: 3 K-d Tree Partitioning

Their purpose is always to hierarchically decompose space into a relatively small number of cells such that no cell contains too many input objects. This provides a fast way to access any input object by position. We traverse down the hierarchy until we find the cell containing the object. Typical algorithms construct k-d trees by partitioning point sets recursively along with different dimensions. Each node in the tree is defined by a plane through one of the dimensions that partitions the set of points into left/right (or up/down) sets, each with half the points of the parent node. These children are again partitioned into equal halves, using planes through a different dimension. Partitioning stops after log n levels, with each point in its own leaf cell [17].

i) K-d Tree Algorithm

The k-d tree is a binary tree in which every node is a k-dimensional point. Every non-leaf node can be thought of as implicitly generating a splitting hyper-plane that divides the space into two parts, known as half-spaces. Points to the left of this hyper-plane represent the left subtree of that node and points right of the hyper-plane are represented by the right subtree. The hyper-plane direction is chosen in the following way: every node in the tree is associated with one of the k dimensions, with the hyper-plane perpendicular to that dimension's axis. So, for example, if for a particular split the "x" axis is chosen, all points in the subtree with a smaller "x" value than the node will appear in the left subtree and all points with larger "x" value will be in the right subtree. In such a case, the hyper-plane would be set by the x-value of the point, and its normal would be the unit x-axis [18].

The nearest neighbor search (NN) algorithm aims to find the point in the tree that is nearest to a given input point. This search can be done efficiently by using the tree properties to quickly eliminate large portions of the search space. Searching for a nearest neighbor in a *k*-d tree proceeds as follows:

1. Starting with the root node, the algorithm moves down the tree recursively, in the same way that it would if the search point were being inserted (i.e. it goes left or right depending on whether the point is less than or greater than the current node in the split dimension).
2. Once the algorithm reaches a leaf node, it saves that node point as the "current best".
3. The algorithm unwinds the recursion of the tree, performing the following steps at each node:

1. If the current node is closer than the current best, then it becomes the current best.
2. The algorithm checks whether there could be any points on the other side of the splitting plane that are closer to the search point than the current best. In concept, this is done by intersecting the splitting hyper-plane with a hyper-sphere around the search point that has a radius equal to the current nearest distance. Since the hyper-planes are all axis-aligned this is implemented as a simple comparison to see whether the difference between the splitting coordinate of the search point and current node is less than the distance (overall coordinates) from the search point to the current best.
1. If the hyper-sphere crosses the plane, there could be nearer points on the other side of the plane, so the algorithm must move down the other branch of the tree from the current node looking for closer points, following the same recursive process as the entire search.
2. If the hyper-sphere doesn't intersect the splitting plane, then the algorithm continues walking up the tree, and the entire branch on the other side of that node is eliminated.

When the algorithm finishes this process for the root node, then the search is complete.

G. Brute Force

Approximate k-nearest neighbour (kNN) search using a brute force approach as well as with the help of the k-d tree will be used to reach of the main objective of this research (i.e. to speed-up K-nearest neighbour searches). Brute-force search or exhaustive search is a very general problem-solving technique that consists of systematically enumerating all possible candidates for the solution and checking whether each candidate satisfies the problem's statement [19].

1. Compute all the distances between the query point and reference points.
2. Sort the computed distances.
3. Select the k reference points with the smallest distances.
4. Classification vote by k nearest objects.
5. Repeat steps (1 to 4) for all query points.

II. EXPERIMENTAL RESULTS AND OUTPUTS

In this paper, presented the experimental results was conducted for computing distance and execution time for nearest neighbours, with number of data set points assuming are organized in 2 dimensions. In first experiment, as shown in figure 4. Analyze the brute force

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 1, January 2014

algorithm that computes the distance and execution time between randomly generated number of data set points. Number of data set points is 11. Enter range one number after the another: (min, max) numbers are 1 to 10. Randomly data set of 11 points is generated and the distance in nearest neighbour & execution time (in microseconds) is computed by brute force nearest neighbour algorithm. In second experiment, as shown in figure 5. We presented an analyzed k-d tree algorithm that computes the distance and execution time between number of data set points. Number of data set points in file is 10. We entered two co-ordinates of the point and the distance between these points and execution time (in microseconds) is computed by the k-d tree nearest neighbour algorithm.

```
How do you want to find nearest neighbor ?
1. By Generating Random numbers
2. By Entering points

Enter your choice:-
1
Enter no. of Points:
11
Enter range one no. after the another: (min,max) -
1
10
=======================================
       Generated 11 random points
=======================================
[(5.85, 4.17), (5.42, 7.97), (9.13, 7.95), (8.13, 3.28),
 (2.87, 9.34), (6.6, 7.56), (3.45, 7.41), (6.69, 4.07),
 (3.38, 3.18), (1.77, 2.6), (9.37, 2.91)]
Brute force Nearest Neighbour with distance:
(5.85, 4.17)-(6.69, 4.07) : 0.8459314393022648
-------------------------------------------------
Do you want to run again? (y/n):
Brute force Algorithm Execution Time: 7188ms
```

Figure: 4 Brute-force output with distance and execution time

```
Console
KDTNearest [Java Application] C:\Program Files\Java\jre7\bin\javaw.exe (Nov 22, 2013 8:36:44
================================================
Welcome to Kd-Tree Nearest Neighbor Search
================================================


How do you want to find nearest neighbor ?
1. From file points
2. Input points

Enter your choice :-
1
 no. of points in file :10
Enter the co-ordinates of the point: (one after the other)
5.85
4.17
Kd tree Nearest Neighbour  :(6.69 , 4.07)
Distance between these points : 0.8459314393022648
Do you want to run again ? (y/n) :
Kd-tree Algorithm Execution Time : 4781ms
```

Figure: 5 K-d Tree output with distance and execution time

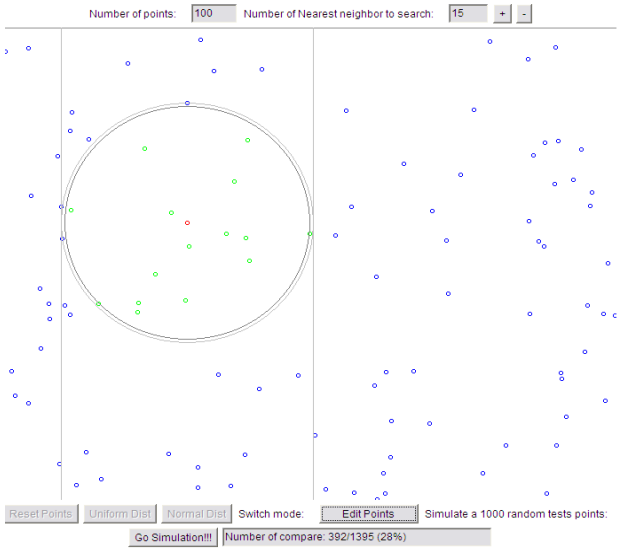


Figure: 6

In this figure, we consider N randomly generated points shown as blue small circles. Then we consider a point from all randomly generated points and find K nearest neighbours in accordance with that point shown as red small circle. We define the value of K. All the nearest neighbours in accordance with that point shown as green small circles. In this case, we use number of points is 100 and search 15 nearest neighbour. Switch mode used for two purposes edit points and select nearest neighbour points.

A. *Table*

Given table show the distance and execution time of both the algorithms when number of data set points are computed. The execution time required by both approaches was different. Distance found by both approaches with the same points. The K-d Tree algorithm needs less number of seconds in comparing with brute-force approach.

Table I: Comparison Of Brute-Force And K-D Tree Algorithm

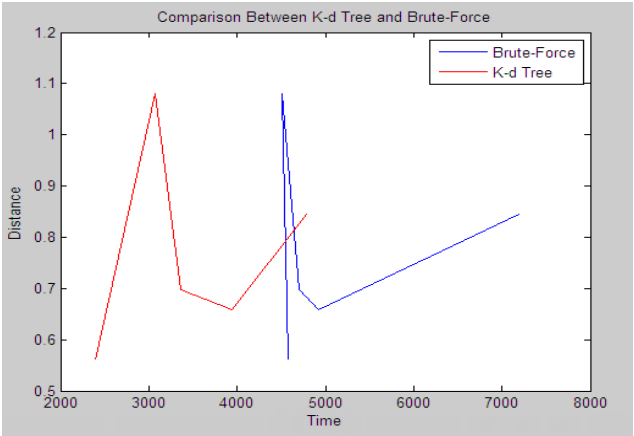
Distance	Brute-force Time	K-d tree Time
0.845931	7188(ms)	4781(ms)
0.658027	4923(ms)	3938(ms)
0.697710	4707(ms)	3359(ms)
1.080046	4512(ms)	3078(ms)
0.560892	4574(ms)	2391(ms)

B. Graphical Representation

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

International Journal of Advanced Research in Computer and Communication Engineering
Vol. 3, Issue 1, January 2014

The experiments have been performed on a 1.73 GHz PC, with Intel Pentium Dual-Core and 1 GB memory. The experiment was conducted for computing distance and execution time for nearest neighbours, with number of data set points assuming are organized in 2 dimensions. The experiment has been performed to compare the performance of both, K-d tree and brute-force, for data set points in a 2-dimentional space. The execution time required by both approaches was different. Distance found by both approaches with the same points. Better performance was achieved when the k-d tree was used for larger number of data sets for nearest neighbours. It shows the execution time and distance required by K-d tree and Brute-force when number of data set points are computed. However, number of data set points for nearest neighbours, better performance was observed using the K-d tree.



Graph1: K-d Tree vs. Brute-Force Performance

The K-d Tree algorithm needs less number of seconds in comparing with brute-force especially when using a large volumes of data in a 2-dimension space.

III. CONCLUSION

The use of the approximate k-nearest neighbour with k-d Tree data structure and comparing its performance to the brute-force approach. The work is done between two techniques and select the best one. The results of the work performed in this paper revealed better performance using the k-d Tree, compared to the traditional brute-force approach. The aim of the algorithm is to make faster, more accurate and efficient data structure primarily depends on a particular data set. It can be further expanded as by changing the k-d tree traversal technique. We have proposed a new modified traversal technique for k-d tree. From the implementation of K-d tree and Brute-force algorithms we now can conclude that methods in K-d tree are comparatively faster than Brute-force.

ACKNOWLEDGEMENT

I would like to thank my guide Er. Namita Kakkar, Assistant Professor in Department of Computer Science and Engineering at R.B.I.E.B.T. (Kharar), Punjab, India for motivating me to do research work on the topic Comparison of Brute-force and K-d tree algorithm. I would also like to thank my mother, my father and my siblings for their continuous support, cooperation, and guidance throughout my M.Tech work. Moreover I would also like to thank my Professors who were always there at the need of the hour and provided with all the help and facilities, which I required, for my thesis work.

REFERENCES

[1] T. M. Cover and P. E. Hart, (1967) "Nearest Neighbor Pattern Classification", IEEE Trans. Inform. Theory, Vol. IT-13, pp 21-27.
[2] Christian Böhm,(2002) "Powerful Database Primitives to Support High Performance Data Mining", Tutorial, IEEE Int. Conf. on Data Mining.
[3] Anoop Jain , Parag Sarda , & Jayant R. Haritsa, (2003) "Providing Diversity in K-Nearest Neighbour Query", Tech. Report TR-2003-04.
[4] R. F Sproull, (1991) "Refinements to Nearest Neighbor Searching", Technical Report, International Computer Science, ACM (18) 9, pp 507-517.
[5] T. Liu, A. W. Moore, A. Gray, (2006) "New Algorithms for Efficient High Dimensional Non- Parametric Classification", Journal of Machine Learning Research, pp 1135-1158.
[6] Ian H. Witten and Eibe Frank, Data Mining- Practical Machine Learning Tools and Techniques- second Edition.
[7] Fayyad U.M., Piatetsky-shapiro G., Smyth P.,Uthurusamy R.(Eds.): Advances in Knowledge Discovery and Data Mining. Menlo Park, CA:AAAI Press/ The MIT Press, 1996.
[8] Paul E. Black (ed.), entry for data structure in Dictionary of Algorithmsand Data Structures. U.S National Institute of Standards and Technology. 15 December 2004. Online version Accessed May 21, 2009.
[9] Parlante, Nick (2001). "Binary Trees" CS Education Library. Stanford University.
[10] Gilberg, R.; Forouzan, B. (2001), "8", Data Structures: A Pseudocode Approach With C++, Pacific Grove, CA: Brooks/Cole, p. 339, ISBN 0-534-95216-X.
[11] Andrews, L.. A template for the nearest neighbor problem. Journal, vol. 19, no 11 (November 2001), 40 - 49, 2001, ISSN:1075-2838, www.ddj.com/ architect/ 184401449
[12] Nitin Bhatia, Vandana, (2010) "Survey of Nearest Neighbor Techniques" International Journal of Computer Science and Information Security, Vol. 8, No. 2
[13] Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). Classification and regression trees. Monterey, CA: Wadsworth & Brooks/Cole Advanced Books & Software.
[14] SAHARKIZ., TEKNOMO, KARDI.,(FEB 2009). K-NEAREST NEIGHBOR ALGORITHM, TUTORIAL. HTTP:\\PEOPLE. REVOLEDU. COM\KARDI\ TUTORIAL\KNN.
[15] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman and A. Wu, (1998) "An optimal algorithm for approximate nearest neighbor searching", Journal of the ACM, 45(6):891-923.
[16] William R. Mark, Gordon Stoll, (2006) "Fast kd-tree Construction with an Adaptive Error-Bounded Heuristic", Warren Hunt, IEEE Symposium on Interactive Ray Tracing.
[17] Steven S. Skiena, (2010) "The Algorithm Design Manual", 2nd Edition, Stony Brook, NY 11794 - 4400.
[18] J.L. Bentley, (1975) "Multidimensional binary search trees used for associative searching", Comm. ACM, 18(9): 509 517.
[19] Stoimen., Stoimen's web log: Computer Algorithms: Brute Force String Matching. March 2012.
[20] R. Panigrahy. Entropy based nearest neighbor search in high dimensions. In SODA '06: Proceedings of the seventeenth annual

ISSN (Print) : 2319-5940
ISSN (Online) : 2278-1021

ACM-SIAM symposium on Discrete algorithm (Miami, FL, 2006), pages 1186–1195. ACM Press, New York, NY, 2006.

[21] Böhm C., Braunmüller B., Krebs F., Kriegel H.-P.: Epsilon Grid Order: An Algorithm for the Similarity Join on Massive High-Dimensional Data, ACM SIGMOD Int. Conf. on Management of Data, 2001.

[22] JOHN PIETER Authenticated Multistep Nearest Neighbour Search in Data mining International Journal of Communications and Engineering Volume 01– No., Issue: 02 March2012.

[23] David Hand, Heikki Mannila, and Padhraic Smyth, 2001, Principles of Data Mining, MIT Press, Cambridge, MA, 2001.

[24] Weinberger, K.Q., Saul, L.K.: Distance Metric Learning for Large Margin Nearest Neighbor Classification. Journal of machine Learning Research 10, 207–244 (2009).

[25] A.N.Pathak, Manu Sehgal, Divya Christopher: A Study on Selective Data Mining Algorithms, IJCSI International Journal of Computer Science Issues, Vol. 8, Issue 2, March 2011.

[26] Perrizo, M., Denton, and Wang, L. (2005) 'An efficient weighted nearest neighbour classifier using vertical data representation',, International Journal of Business Intelligence and Data Mining, Vol. 1, No. 1, pp.65–87.

[27] David M. Mount, Analysis of Approximate Nearest Neighbor Searching with Clustered Point Sets, DIMACS Series in Discrete Mathematics and Theoretical Computer Science 1991.

[28] Jagan Sankaranarayanan, Hanan Samet, Amitabh Varshney: A fast all nearest neighbor algorithm for applications involving large point-clouds, Computers & Graphics 31 (2007) 157.174. (Best Journal Paper of 2007Award).

[29] Madhuri V. Joseph Lipsa Sadath Vanaja Rajan: Data Mining: A Comparative Study on Various Techniques and Methods. International Journal of Advanced Research in Computer Science and Software Engineering 3(2), February - 2013, pp. 106-113.

[30] Barbara D. (ed.) Special Issue on Mining of Large Datasets; IEEE Data Engineering Bulletin, 21(1), 1998.