Finding Similarity in Data points using K-d trees

May 27, 2022

Subham Subhasis Sahoo (2020CSB1317) ,
Pooja Goyal (2020CSB1108) ,
Jayesh Bhojawat (2020MCB1239)

Instructor:
Dr. Anil Shukla

Teaching Assistant:
Sravanthi Chede

Summary: In this project we tried to predict the similarity of data points using nearest neighbour algorithm which is implemented through the K-d tree data structure making it time efficient.

---

1. Introduction

A K-D Tree is a binary search tree where data in each node is a K-Dimensional point in space.It is a space partitioning data structure for organizing points in a K-Dimensional space. Every node in the tree represents a point in space. Every non leaf node in the KD- tree indicates along which dimension the space was divided by the node; also each subspace can be recursively divided in the same way. All subspaces are divided into two parts, left subspace and right subspace. Procedure to construct a K-D tree is to recursively divide the space in 2 parts along the axis that has the widest spread.

K Dimensional data set on which the tree is created represents a partition of the Kdimensional space formed by the K-dimensional data set. Every node in the tree corresponds to a K-dimensional hyper-rectangle area.

The properties of KD tree is quite similar to binary tree in certain dimension such as :

[1] If the left subtree is not empty then the value of all nodes on the left subtree are not greater than the value of its root node.
[2] If the right subtree is not empty then the value of all nodes on the left subtree are not greater than the value of its root node.
[3] Its left and right subtrees are also binary sort trees.

2. Time Complexity Analysis

To find the k nearest neighbours using brute force, the distance between every point in the data set is calculated using the Euclidean equation:
The Euclidean distance between a and b (a and b are n dimensional vectors) is defined as follows:

$$\boldsymbol{d(a,b)} = \sqrt{(x_{a1} - x_{b1})^2 + (x_{a2} - x_{b2})^2 + (x_{a3} - x_{b3})^2 + ... + (x_{an} - x_{bn})^2} \tag{1a}$$

For every point, distance between it and all the other points can be calculated in O(n) time. Now sorting can be done on the basis of distances obtained in worst case O(n*n) time and now the k net neighbours can be obtained in O(k) time.
So for n points, the time complexity would be n*O(n^2)

Hence the overall time complexity to find K Nearest Neighbours using brute force is:

$$O(n^3) \tag{2a}$$

Now, since the sorting can be done in O(nlog n) time, the above algorithm can be improved to:

$$O(n^2 logn) \tag{3a}$$

However, if implemented using KD Tree, the time complexity can be reduced to:

$$O(klogn) \tag{4a}$$

This is because searching a node in KD Tree requires O(log n) time and node can be pushed or popped in stack in O(1) time. While searching ,we have to compare the datapoints which are arrays of k size at every level to see if the node matches the point we are searching for.
So the complexity becomes O(k*logn).

3. Figures and Algorithms

3.1. Figures

KD-Tree decomposition for the example data set with the following data points : (2,3), (5,4), (9,6), (4,7), (8,1), (7,2)
The generated tree is shown below.

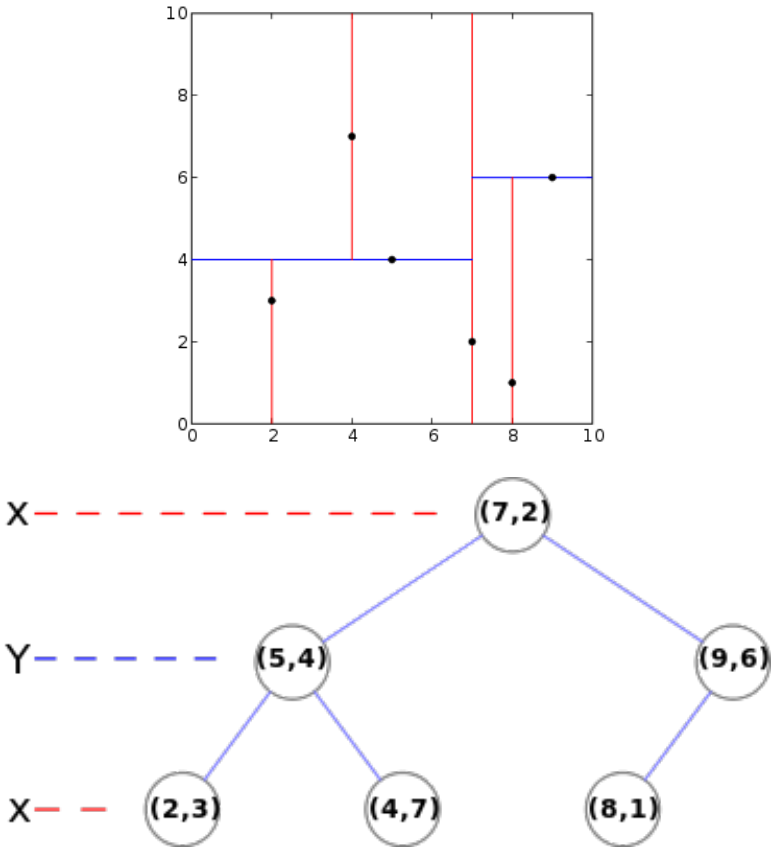

Figure 1: The resulting KD - tree.

This gives an idea about how the kd tree data structure stores data in a spatial arrangement in the k-dimensional space.(here k=2)

3.2. Algorithms

Algorithm 1 INSERT A NODE IN KD TREE

```
insert (kdnode root,point[ ],depth)
int currdepth=depth % k
if (root ==NULL)
    root=create kdnode
if(point[currdepth]<root->point[currdepth])
    root->left=insert(root->left,point[ ],depth+1)
else
    root->right=insert(root->right,point[ ],depth+1)
```

Algorithm 2 SEARCHING IN KD TREE

```
search(kdtree root,point[ ],depth)
if root ==NULL
    return false
if root->point=point
    return true
int currdepth =depth % k
if (point[currdepth]<root->point[currdepth])
    return search (root->left,point[ ],depth++)
else
    return search(root->right,point[ ],depth++)
```

Algorithm 3 FINDING SIMILARITY IN DATA POINTS FOR A GIVEN POINT

```
start at root
search()
push each datapoint in stack along the search path (pushback(datapoint))
while(number of nearest neighbours –)
    pop();
output K nearest neighbours
```

4. Conclusions

In this project, we optimised the nearest neighbor algorithm using KD trees. The proposed algorithm showed an improvement in search efficiency, reducing its time complexity to O(k*log n). The algorithm is also able to correctly predict the outcome of data points with 80% accuracy.

5. Bibliography and citations

We thanks to Sravanthi Chede mam for guiding us in the implementaion.Below is the list of some of the resources through which we have collected information.
[1] https://developpaper.com/kd-tree-implementation-of-k-nearest-neighbor-algorithms/
[2] https://www.analyticsvidhya.com/blog/2017/11/information-retrieval-using-kdtree/
[3] CLRS BOOK