**Algorithms & Analysis Assignment 1**

Student 1: name & student number

Student 2: name & student number

We certify that this is all our group’s original work. If we took any parts from elsewhere, then they were non-essential parts of the assignment, and they are clearly attributed in our submission. We will show that we agree to this honour code by typing ``Yes": YES.

**Experimental Setup**

Describe briefly how you generated your data?

The data is generated in different groups different in sample size N, neighbour size K and query size M.

The sample size N are consisted of three groups: small, middle and large (1000, 10000, 100000 samples).

The neighbour size K are consisted of three groups: {1, 3, 7}.

The query size Mare consisted of three groups: {10, 100, 1000}

First we generated the sample data in 1000, 10000 and 100000 size. The data is generate by uniform random functions in python, Numpy.random . The latitude is in range of (-90,90) and longitude is in range of (-180,180). The category data is random selected and the id is generated as N grows.

After the sample data are generated, we import the data above as pandas.DataFrame, and sample M records from the it. The M records are used to generate command input file. Similar method is applied to generate dynamic data sample, where M stands for M records to ADD, DELETE and SEARCH.

In dynamic point scenario, firstly N data records are random generated. Then the M ADD data are generated and contacted with the N data records set. Among (N+M) records, N records are random sampled to generate DELETE records. Lastly, M SEARCH records are random generated.

The idea that to design the sample data is, we want to discover how the three different factors (N, K, M) would affect the performance of the models. In reality, the number of sample N is one of the most important factors. Intuitively the time on computation is proportional to N. Also, we are interested in how K will impact the model performance. Based on theory and the data structure, the K plays a rather small role in performance. Finally we want to discover the influence of the number of queries M.

We set the timing loggers at each Interface of NearestNeigh. Total processing time of total execution and individual method were recorded. Each combination of (N, K, M) in two scenario( k-NN and dynamic points) are calculated. Each case we ran 5 times and the average total execution time (build, add, search, delete) are calculated. The details is in Fig 1 and Fig 2 .

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Model | M=10 | | | M=100 | | | M=1000 | | |
|  |  | K=1 | K=3 | K=7 | K=1 | K=3 | K=7 | K=1 | K=3 | K=7 |
| N=1000 | KdTree | 139.8 | 142.2 | 139.0 | 148.4 | 162.8 | 164.0 | 188.2 | 231.5 | 273.8 |
| Naive | 133.4 | 134.2 | 137.4 | 191.8 | 193.4 | 185 | 488.0 | 488.0 | 490.2 |
| N=10000 | KdTree | 303.4 | 320 | 307 | 309.6 | 318.2 | 341 | 357.4 | 402.8 | 464.8 |
| Naive | 278.8 | 377 | 273 | 445.2 | 464.4 | 466.7 | 1690.4 | 1688.4 | 1699.6 |
| N=100000 | KdTree | 1266.6 | 1243.2 | 1247.6 | 1241.6 | 1259.6 | 1266.6 | 1303.8 | 1348.4 | 1417.6 |
| Naive | 882.6 | 903.2 | 907.5 | 2016.6 | 2003.2 | 2026.6 | 11175.2 | 11170 | 11242 |

Fig 1 The total execution time of KdTree and Naïve model. N=sample number, M= search number, K =neighbour query number. Time in milesecond.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| K=7 | KD Total  Time | KD Build-Index(avg) | KD Search(avg per) | Naïve  Total  Time | Naïve Build-Index(avg) | Naïve Search(avg per) |
| N=10000  M=10 | 307 | 127 | 0.1 | 273 | 0 | 2.7 |
| N=10000  M=100 | 341 | 128 | 0.1 | 466.7 | 0 | 4.67 |
| N=10000  M=1000 | 464.8 | 128 | 0.1 | 1699.6 | 0 | 1.70 |
| N=100000  M=100 | 1266.6 | 134 | 0.1 | 2026.6 | 0 | 20.26 |

Fig 2 The average execution time of Build and Search in KD-tree and Naïve.

**Evaluation**

Scenario 1 (k-nearest neighbour searches)

1. K. We found that k, the number of nearest neighbours almost makes little impact on brute force performance and KD-tree (see Figure 1). At first, we hypothesise the reason for this is that both models can performance well when k increased. For example, we use a max heap to keep Top-K minimum points searched so far, which is effective for get k nearest neighbour. However, when we examine the prune algorithm in KD-tree, it comes clear that when K>N/2, the branches could not be pruned. To naive approach, since data is sorted, the K value is not important.

Thus, we can say when k is considerably large(K>N/2), the KD-tree may be out-performance by brutal-force. Otherwise the KD-tree could be working well since we can prune useless branch.

1. N. We found that N, the number of sample records, is proportional to the calculation time of both models. From Fig 2, there are comparison about the Build Index time and Search Time to KD-tree and Naïve approach. We can conclude that, the query average per time is growing faster in naïve approach, and changes little in KD tree. The most overhead in KD-tree process is building index while there’s no building cost in naïve method.

In fact, according to the time complexity analysis, Brute force query time grows as O(DN).

KD tree query time grows as O(DLogN). [1]

1. M. From Fig 1 and Fig 2, we found the brutal force query time grows fast with M. The total execution time of brutal force grows as O(DNM) because each search requires a re-sort. In contrary, the query time of KD-tree doesn’t grow much with M, because we don’t need to reconstruct the KD-tree again each query.

Scenario 2 (Dynamic points set)

|  |  |  |  |
| --- | --- | --- | --- |
| K=5 | M=10 | M=100 | M=1000 |
| N=1000 | 145 | 191 | 367 |
| N=10000 | 342 | 375 | 479 |
| N=100000 | 1169 | 1208 | 1340 |

Fig 3 The dynamic points set’s performance on Kd-tree.

As we performed more adds and equivalent number of deletions to the kd-tree, we found that when the change ratio M/N is not big enough, the performance of KD-tree is not degraded. However, if the change ratio is larger enough (e.g. 1000/1000, almost every point could be changed in worst case. When N=1000 and M=1000, the total execution time is 367 in average and is larger than unchanged group(273.8). It could be caused by the large portion of tree node updates , leading to unbalanced or unstructured tree data structure.

**Recommendation**

Conclude as above, we could say

1. When M is smaller(M<1000), the brutal force may achieve good performance. In contrast, when M is growing faster, the overhead of search per query should be considered and the KD-tree should be selected.
2. When M is larger (M is at least the same order as N), the building time of KD-tree is amortized by the number of queries. The Index building time grows with O(ND). So the KD-tree performance better.
3. When N is small (N<30), brutal force can be more efficient since N is comparable with Log(N). When N is larger, the KD-tree should increase it’s performance.
4. When K grows the KD-tree becomes slower because it cannot utilize the prune to achieve good performance, especially when(K>=N/2). The K is not largely related to brutal-force’s performance.

Given the parameters, we can summarize:

1. K>=N/2 or M is small or N is small: Brutal force
2. Otherwise, choose KD tree.

**Further Discussion**

The number of dimensions is not discussed in this article since our research area is focused on 2-dimension points. In fact, when D grows(D>20), KD-tree performance badly as other tree data-structure, which is so called curse of dimensionality [2]. More generally, the data can be stored by a 2-d array, to represent multiple Dimension data. The array-based approach is more widely used.

To construct the KD-tree, we use a sorted point list to find median point and split on cutting dimension. Copies are made to store information which leads to extra memory usage. There’s a in-place quick select algorithm to swap the points split by cutting dimension, which is more time efficient. [3]

The cycling through cutting dimensions may not be the best approach. In a 3-d dimension space, when points are located in same z-coordinate plane, it is pointless to split the space by switching cutting dimension in turn and us z-axis. Max variance in dimension could be a better way.

[1] https://scikit-learn.org/stable/modules/neighbors.html#nearest-neighbor-algorithms

[2] Wikipedia articles on Curse of dimensionality, [https://en.wikipedia.org/wiki/Curse\_of\_dimensionality]

[3] CMU Kd-tree, [https://www.cs.cmu.edu/~ckingsf/bioinfo-lectures/kdtrees.pdf]