Central Washington University

Advance Algorithm

Winter 2020

Project 1: k-Nearest Neighbor Search

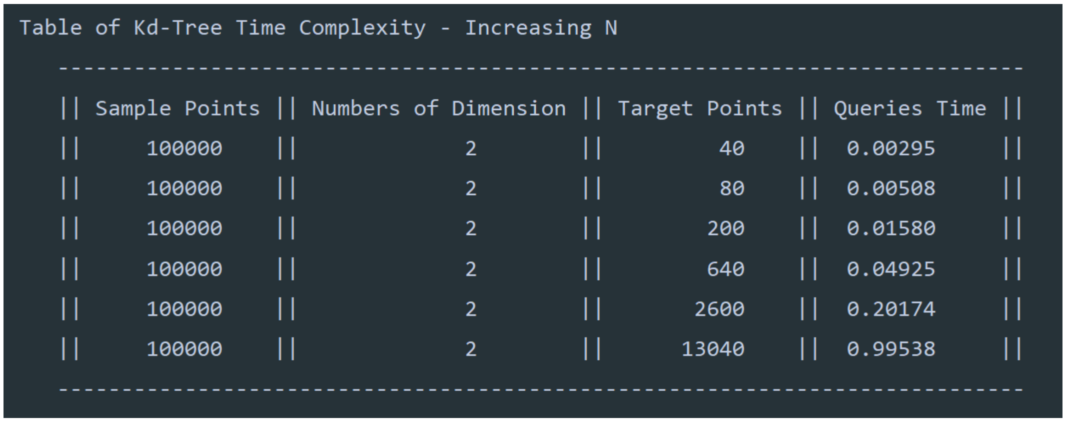
Student: Tin Tan Nguyen
Email: ntin@cwu.edu

January 30, 2020

1 Time Complexity Analysis - Increasing N

This section will discuss run time of k-nearest neighbor search using Python - scipy.spatial.kdtree package when increasing numbers of nearest neighbor.

- **Increasing N number of nearest neighbors:**



```
Table of Kd-Tree Time Complexity - Increasing N
   -----------------------------------------------------------------------
   || Sample Points || Numbers of Dimension || Target Points || Queries Time ||
   ||     100000    ||           2          ||            40 ||  0.00295     ||
   ||     100000    ||           2          ||            80 ||  0.00508     ||
   ||     100000    ||           2          ||           200 ||  0.01580     ||
   ||     100000    ||           2          ||           640 ||  0.04925     ||
   ||     100000    ||           2          ||          2600 ||  0.20174     ||
   ||     100000    ||           2          ||         13040 ||  0.99538     ||
   -----------------------------------------------------------------------
```

Figure 1: Time complexity when increasing N

- **Time vs N-increasing Plot:**

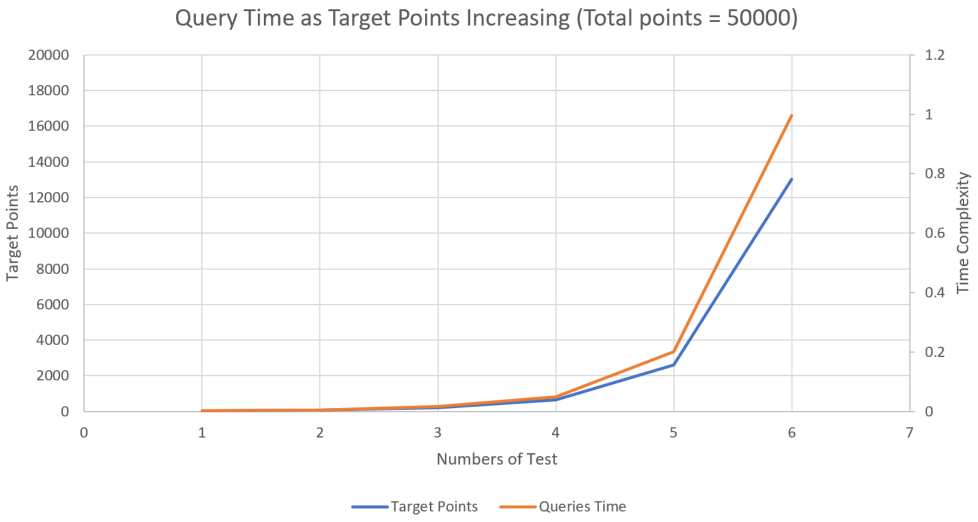


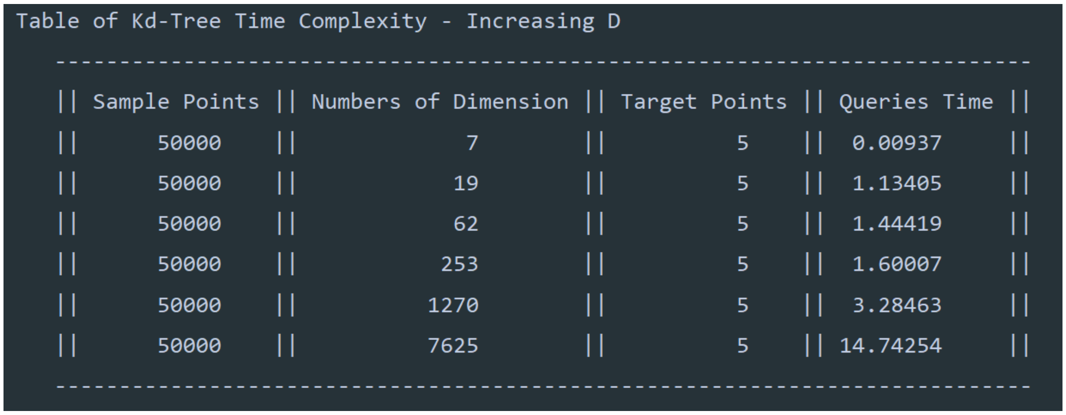
Figure 2: Time complexity as N increasing

The relationship between run-time and N is almost linear by looking at the chart above (Time-complexity <= TotalPoints)

2 Time Complexity Analysis - Increasing D

This section will discuss run time of k-nearest neighbor search using Python - scipy.spatial.kdtree package when increasing dimension of nearest neighbor.

- **Increasing number of dimensions (N = 5):**



```
Table of Kd-Tree Time Complexity - Increasing D

   ----------------------------------------------------------------------

   || Sample Points || Numbers of Dimension || Target Points || Queries Time ||
   ||     50000     ||           7          ||        5      ||  0.00937     ||
   ||     50000     ||          19          ||        5      ||  1.13405     ||
   ||     50000     ||          62          ||        5      ||  1.44419     ||
   ||     50000     ||         253          ||        5      ||  1.60007     ||
   ||     50000     ||        1270          ||        5      ||  3.28463     ||
   ||     50000     ||        7625          ||        5      || 14.74254     ||

   ----------------------------------------------------------------------
```

Figure 3: Time complexity when increasing D

- **Time vs D-increasing Plot (NN = 5, Total Points = 50000):**

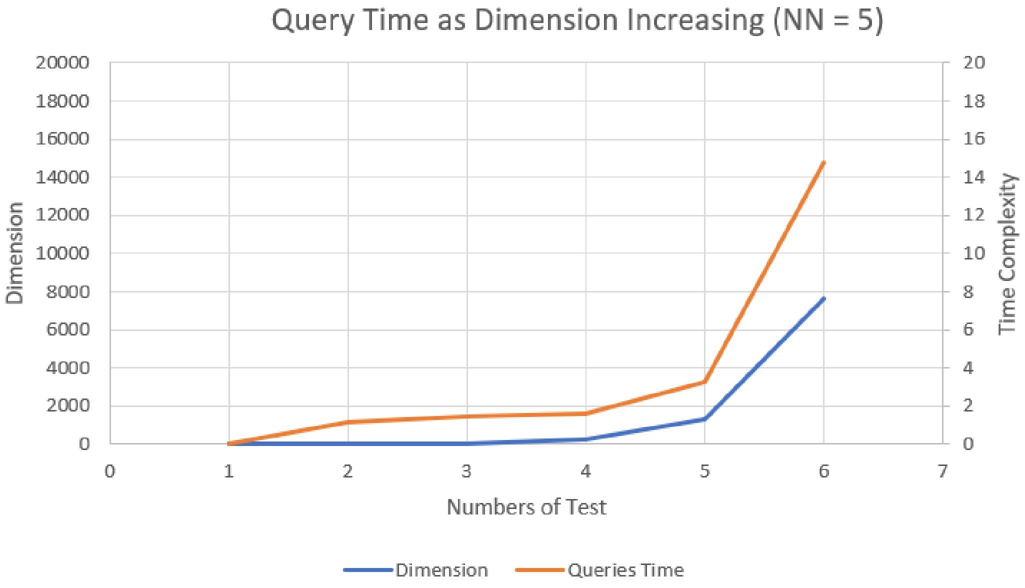


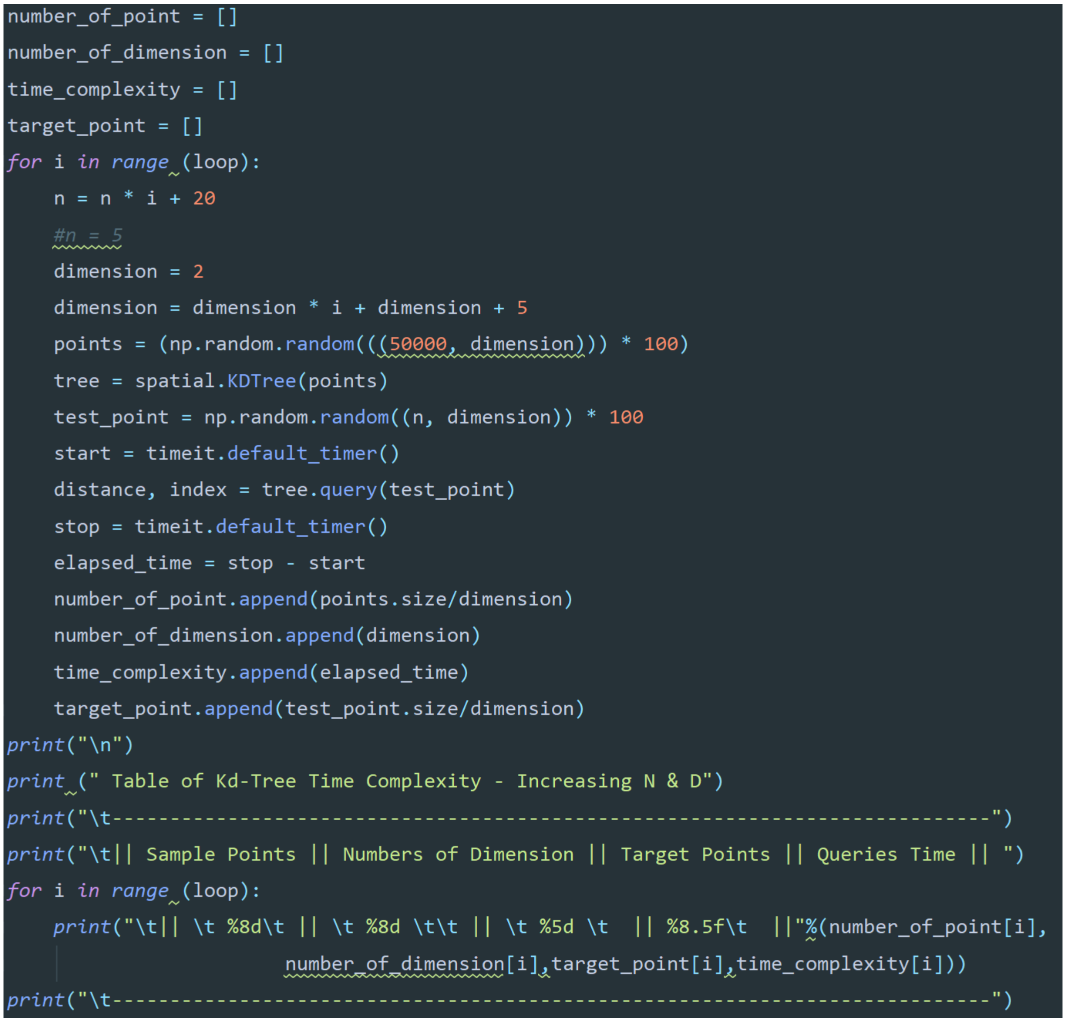
Figure 4: Time complexity as D increasing

The relationship between run-time and D is not a linear relationship by looking at the chart above (Time-complexity >= TotalPoints and similar to log(TotalPoints)* TotalPoints.

3 Conclusion

Time complexity of k-nearest neighbor algorithm is n times based on this project when increasing the NN numbers and about lg(n)n when increasing the dimension of NN.

4 Python Code



```python
number_of_point = []
number_of_dimension = []
time_complexity = []
target_point = []
for i in range (loop):
    n = n * i + 20
    #n = 5
    dimension = 2
    dimension = dimension * i + dimension + 5
    points = (np.random.random(((50000, dimension))) * 100)
    tree = spatial.KDTree(points)
    test_point = np.random.random((n, dimension)) * 100
    start = timeit.default_timer()
    distance, index = tree.query(test_point)
    stop = timeit.default_timer()
    elapsed_time = stop - start
    number_of_point.append(points.size/dimension)
    number_of_dimension.append(dimension)
    time_complexity.append(elapsed_time)
    target_point.append(test_point.size/dimension)
print("\n")
print (" Table of Kd-Tree Time Complexity - Increasing N & D")
print("\t----------------------------------------------------------------------------")
print("\t|| Sample Points || Numbers of Dimension || Target Points || Queries Time || ")
for i in range (loop):
    print("\t|| \t %8d\t || \t %8d \t\t || \t %5d \t  || %8.5f\t  ||"%(number_of_point[i],
                        number_of_dimension[i],target_point[i],time_complexity[i]))
print("\t----------------------------------------------------------------------------")
```

Figure 5: Python Source Code