430.217

Final Project: Implementation of DBSCAN

2012-11598 민두기

1. Implementation of kd-tree

Kd-tree 구현은 [1]을 참고하였다. 실제 좌표를 가진 node 들은 leaf 에만 존재하도록 구현하였다. 기본적인 구현은 다음과 같다.

Constructor 에서 float ** points 들을 받아오면 class Coordinate 를 node 로 사용하여 kd-tree 를 build 한다. Sorting 과 partitioning 을 쉽게 하기 위해 Coordinate ** dataset 이라는 멤버를 두어 Coordinate *들을 array 로 다루었다. 이 때 Build_KD_Tree 함수를 slave 로 사용하여 build 하는데 현재 axis 에 해당하는 hyper plane 을 구하기 위해서 Time complexity 가 O(n)인 median of median algorithm 을 사용하였다[2]. 좀더 적절한 hyper plane 을 구하기 위해서 split point 는 median node 와 그 바로 오른쪽에 있는 node 의 현재 axis point 의 값이 바뀌는 지점을 선택하였고 만약 median node 가 가장 오른쪽에 존재한다면 이 때 duplicate node 인지 검사를 하게 된다(split 이 안일어 난다는 의미이므로).

```
//find real split point that has median data
while(left + (right - left)/2 + i <= right)
{
    if(dataset[left+(right-left)/2]->point[depth%dimension] < dataset[left +
        (right - left)/2 + i]->point[depth%dimension])
        break;
    i++;
}
//if split point is right ---> no split
```

이 때 duplicate node 검사를 하는 이유는 split 이 안일어나는 경우는 현재 range 에 노드가 몇개 안남은 상황이라 생각하였고, 이 때 duplicate 검사를 하여 duplicate node 를 discard 시키는게 time complexity 측면에서 적절하다고 생각하였다. Array 두개를 index 하나 하나 비교해서 하나라도 다르면 false 를 return 하는 same_array 함수를 두어서 현재 range 에 존재하는 모든 node 들을 살피게 하였다. 이렇게 split 된 point 들을 다시 Bulid_KD_Tree 함수의 input 으로 두어 recursive 하게 build 가 이루어진다. 마지막으로 Coordinate 의 member function 인 set_lower_and_upper_bound 라는 함수를 call 해 현재 hyper plane vertex 의 upper bound 와 lower bound 를 update 해준다. Upper bound 와 lower bound 는 현재 node 의 아래에 있는 leaf 들이 가지고 있는 data 들의 range 를 나타낸다. 이 과정은 time complexity 의 계수를 증가시키겠지만 나중에 search 를 빠르게 하도록 도와준다.

Member function 인 getNeighbor()는 Coordinate 의 member function 인 serach 의 master 로 slave 인 search 함수를 불러온다.

2. Implementation of Coordinate & CoordinateSet

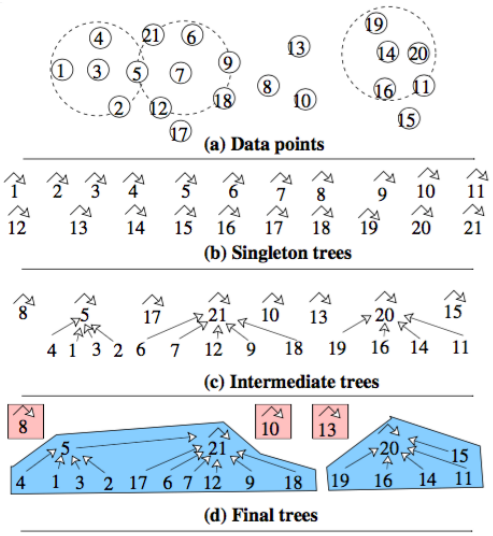
Class Coordinate 는 kd-tree 의 node 로 쓰이기 때문에 member 로 Coordinate *left 와 Coordinate *right 를 가지고 있으며 실제 좌표를 저장하기 위한 float *point 가 존재한다. 또한 internal vertex 를 위해서 현재 hyper plane 의 axis 와 그 axisdata 를 member 가지고 있다.

Member function 중 search 는 input 으로 Coordinate 들을 담을 Coordinate *container, query 포인트인 Coordinate *query, 그리고 반경 float epsilon 을 받는다. 현재 vertex 의 axis data 와 query 의 axis data 의 차이를 이용해서 search 를 자식 node 중 어디 쪽으로 할지 결정하게 된다. Leaf node 에 도착하게 되면 query 와 leaf node 의 Euclidean distance 을 계산하여 epsilon 보다 작으면 container 로 push 한다.

CoordinateSet 의 member 로는 neighbor 들을 linked list 로 엮기위한 Coordinate * head 가 존재한다. Member function 인 push_element 는 Cooridnate *를 input 으로 받으며 평범한 linked list 의 push front method 이다. Print() 함수의 경우 index 로 sorting 해서 print 해야 하기 때문에 Coordinate **를 만들어서 엮었던 neighbor 들을 pointer array 에 집어넣고 quick sort 를 한다. Quick sort 는 partition 이나 median of median algorithm 을 미리 구현해 놨기 때문에 그대로 사용하여 구현하였다.

3. Implementation of DBSCAN

DBSCAN 의 구현은 [3]을 참고하였다. Parallel DBSCAN 은 아니고 논문의 2 번째 algorithm 을 사용하였다.



Algorithm 2 The disjoint-set data structure based DBSCAN Algorithm (DSDBSCAN). Input: A set of points X, distance eps, and the minimum number of points required to form a cluster, $minpts$. Output: A set of clusters.

```
1:  procedure DSDBSCAN(X, eps, minpts)
2:      for each point x ∈ X do
3:          p(x) ← x
4:      for each point x ∈ X do
5:          N ← GETNEIGHBORS(x, eps)
6:          if |N| ≥ minpts then
7:              mark x as core point
8:              for each point x' ∈ N do
9:                  if x' is a core point then
10:                     UNION(x, x')
11:                 else if x' is not yet member of any cluster then
12:                     mark x' as member of a cluster
13:                     UNION(x, x')
```

Constructor 는 KD_Tree &kd_tree, float epsilon, int minPoints 를 input 으로 받아와서 Coordinate **dataset 을 각각 make set 한다. 그후 kd_tree 의 member function 인 getNeighbor 를 input 으로 dataset 각각을 query 를 넣어 call 한다. Neighbor 의수가 minPoints 보다 크거나 같으면 해당 dataset 은 core 가 되며 Neighbor 들과 UNION 한다. 이 때 만약 Neighbor 가 다른 disjoint set 의 core 나 다른 disjoint set 의 member 가 아닐 때만 UNION 한다. (UNION 이나 FIND_SET 은 교수님의 ppt 그대로 구현하였다. Union by rank 와 path compression 모두 사용하였다.) Member function 인 run()은 모든 dataset 을 받아와 index 로 sorting 한후 각각의 dataset 의 parent 의 ID 를 print 한다. 이때 Find_Set 을 다시 사용하여 path compression 을 해준다.

〔1〕KDTREE 2: Fortran 95 and C++ software to efficiently search for near neighbors in a multi-dimensional Euclidean space , Matthew B. Kennel, University of California, San Diego
〔2〕https://en.wikipedia.org/wiki/Median_of_medians
〔3〕A New Scalable Parallel DBSCAN Algorithm Using the Disjoint-Set Data Structure Md. Mostofa Ali Patwary1, †, Diana Palsetia1, Ankit Agrawal1, Wei-keng Liao1, Fredrik Manne2, Alok Choudhary1 1Northwestern University, Evanston, IL 60208, USA 2University of Bergen, Norway