430.217

Final Project: Implementation of DBSCAN

2012-11598 민두기

1. Implementation of kd-tree

Kd-tree 구현은 [1]을 참고하였다. 실제 좌표를 가진 node들은 leaf에만 존재하도록 구현하였다. 기본적인 구현은 다음과 같다.

Constructor에서 float \*\* points들을 받아오면 class Coordinate를 node로 사용하여 kd-tree를 build한다. Sorting과 partitioning을 쉽게 하기 위해 Coordinate \*\* dataset이라는 멤버를 두어 Coordinate \*들을 array로 다루었다. 이 때 Build\_KD\_Tree 함수를 slave로 사용하여 build 하는데 현재 axis에 해당하는 hyper plane을 구하기 위해서 Time complexity가 O(n)인 median of median algorithm을 사용하였다[2]. 좀더 적절한 hyper plane을 구하기 위해서 split point는 median node와 그 바로 오른쪽에 있는 node의 현재axis point의 값이 바뀌는 지점을 선택하였고 만약 median node가 가장 오른쪽에 존재한다면 이 때 duplicate node인지 검사를 하게 된다(split이 안일어 난다는 의미이므로).

이 때 duplicate node 검사를 하는 이유는 split이 안일어나는 경우는 현재 range에 노드가 몇개 안남은 상황이라 생각하였고, 이 때 duplicate 검사를 하여 duplicate node를 discard시키는게 time complexity측면에서 적절하다고 생각하였다. Array두개를 index하나 하나 비교해서 하나라도 다르면 false를 return 하는 same\_array함수를 두어서 현재 range에 존재하는 모든 node들을 살피게 하였다. 이렇게 split된 point들을 다시 Bulid\_KD\_Tree함수의 input으로 두어 recursive하게 build가 이루어진다.

마지막으로Coordinate의 member function인 set\_lower\_and\_upper\_bound라는 함수를 call 해 현재 hyper plane vertex의 upper bound와 lower bound를 update해준다. Upper bound와 lower bound는 현재 node의 아래에 있는 leaf들이 가지고 있는 data들의 range를 나타낸다. 이 과정은 time complexity의 계수를 증가시키겠지만 나중에 search를 빠르게 하도록 도와준다.

Member function인 getNeighbor()는 Coordinate의 member function인 serach의 master로 slave인 search함수를 불러온다.

1. Implementation of Coordinate & CoordinateSet

Class Coordinate 는 kd-tree의 node로 쓰이기 때문에 member로 Coordinate \*left 와 Coordinate \*right를 가지고 있으며 실제 좌표를 저장하기 위한 float \*point가 존재한다. 또한 internal vertex를 위해서 현재 hyper plane의 axis와 그 axisdata를 member가지고 있다.

Member function중 search는 input으로 Coordinate들을 담을 Coordinate \*container, query포인트인 Coordinate \*query, 그리고 반경 float epsilon을 받는다. 현재 vertex의 axis data 와 query의 axis data의 차이를 이용해서 search를 자식node중 어디 쪽으로 할지 결정하게 된다. Leaf node에 도착하게 되면 query와 leaf node의 Euclidean distance을 계산하여 epsilon 보다 작으면 container로 push 한다.

CoordinateSet의 member로는 neighbor들을 linked list로 엮기위한 Coordinate \* head가 존재한다. Member function인 push\_element는 Cooridnate \*를 input으로 받으며 평범한 linked list의 push front method이다. Print() 함수의 경우 index로 sorting해서 print해야 하기 때문에 Coordinate \*\*를 만들어서 엮었던 neighbor들을 pointer array에 집어넣고 quick sort를 한다. Quick sort는 partition 이나 median of median algorithm을 미리 구현해 놨기 때문에 그대로 사용하여 구현하였다.

1. Implementation of DBSCAN

DBSCAN의 구현은 [3]을 참고하였다. Parallel DBSCAN은 아니고 논문의 2번째 algorithm을 사용하였다.

Constructor는 KD\_Tree &kd\_tree, float epsilon, int minPoints를 input으로 받아와서 Coordinate \*\*dataset을 각각 make set한다. 그후 kd\_tree의 member function 인 getNeighbor를 input으로 dataset각각을 query를 넣어 call한다. Neighbor의수가 minPoints보다 크거나 같으면 해당 dataset은 core가 되며 Neighbor들과 UNION한다. 이 때 만약 Neighbor가 다른 disjoint set의 core나 다른 disjoint set 의 member가 아닐 때만 UNION한다. (UNION이나 FIND\_SET은 교수님의 ppt그대로 구현하였다. Union by rank와 path compression모두 사용하였다.)

Member function인 run()은 모든 dataset을 받아와 index로 sorting한후 각각의 dataset의 parent의 ID를 print한다. 이때 Find\_Set을 다시 사용하여 path compression을 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