밀도 기반 군집화 : 클러스터를 데이터가 높은 밀도로 모여 있는 공간으로 보는 기법

(ex DBSCAN)

# DBSCAN

포인트 클라우드 간의 밀도 정보를 이용하여 군집을 구분 하는 방법 입니다. 점이 세밀하게 몰려 있어서 밀도가 높은 부분을 클러스터링 하는 방식이다. 쉽게 설명하면, 어느점을 기준으로 반경 x내에 점이 n개 이상 있으면 하나의 군집으로 인식하는 방식이다.

장점

* K Means와 같이 클러스터의 수를 정하지 않아도 됨
* 클러스터의 밀도에 따라서 클러스터를 서로 연결하기 때문에 기하학적인 모양을 갖는 군집도 잘 찾을 수 있음

단점

* 많은 연산을 수행하기에 K 평균에 비해 그 속도가 느림
* 반지름과 임계치 설정에 많은 영향을 받는다
* 그리고, 유클리드 제곱거리를 사용하는 모든 데이터 모델의 공통적인 단점인, 'Curse Of dimensionality 또한 존재
  + 이는 2차원이나 3차원 등 차원수가 낮은 데이터세트에는 문제가 되지 않지만,
  + 고차원 데이터세트로 갈수록 필요한 학습 데이터 양이 급증하는 문제점이며, 이 때문에 많은 연산이 필요해진다는 단점이 있다.

A low minPts means it will build more clusters from noise, so don't choose it too small. [How can I choose eps and minPts (two parameters for DBSCAN algorithm) for efficient results?](https://www.researchgate.net/post/How_can_I_choose_eps_and_minPts_two_parameters_for_DBSCAN_algorithm_for_efficient_results)

|  |  |  |
| --- | --- | --- |
| image | image | image |
| p=코어포인트 . 반경안에 점 4개 존재 | P2=경계점(Border point) . 반경안에 점 4개 없음 But, Core point에는 속함 | P4=Noise Point . 반경안에 점 4개 없음 .And, Core point에 속하지 않음 (어느 군집에도 속하지 않는다면 이것은 노이즈 데이터라 하며 제외 시킴) |
| image | image | image |
| P=코어포인트  P3 = 코어데이터 | 두 코어 포인트를 연결하여 하나의 군집으로 처리 | **정리** |

Dbscan Examples

**>>> from** **sklearn.cluster** **import** DBSCAN

**>>> import** **numpy** **as** **np**

**>>>** X = np.array([[1, 2], [2, 2], [2, 3],

**...**  [8, 7], [8, 8], [25, 80]])

**>>>** clustering = DBSCAN(eps=3, min\_samples=2).fit(X)

**>>>** clustering.labels\_

array([ 0, 0, 0, 1, 1, -1])

**>>>** clustering

DBSCAN(eps=3, min\_samples=2)

Euclidean-PCL-Cpp

#include <pcl/ModelCoefficients.h>

#include <pcl/point\_types.h>

#include <pcl/io/pcd\_io.h>

#include <pcl/filters/extract\_indices.h>

#include <pcl/filters/voxel\_grid.h>

#include <pcl/features/normal\_3d.h>

#include <pcl/kdtree/kdtree.h>

#include <pcl/sample\_consensus/method\_types.h>

#include <pcl/sample\_consensus/model\_types.h>

#include <pcl/segmentation/sac\_segmentation.h>

#include <pcl/segmentation/extract\_clusters.h>

// Euclidean Cluster Extraction

// http://pointclouds.org/documentation/tutorials/cluster\_extraction.php#cluster-extraction

int main (int argc, char\*\* argv)

{

pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud (new pcl::PointCloud<pcl::PointXYZRGB>);

pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud\_f (new pcl::PointCloud<pcl::PointXYZRGB>);

// \*.PCD 파일 읽기 (https://raw.githubusercontent.com/adioshun/gitBook\_Tutorial\_PCL/master/Intermediate/sample/RANSAC\_plane\_true.pcd)

pcl::io::loadPCDFile<pcl::PointXYZRGB> ("RANSAC\_plane\_true.pcd", \*cloud);

// 포인트수 출력

std::cout << "PointCloud before filtering has: " << cloud->points.size () << " data points." << std::endl; //\*

// 탐색을 위한 KdTree 오브젝트 생성 //Creating the KdTree object for the search method of the extraction

pcl::search::KdTree<pcl::PointXYZRGB>::Ptr tree (new pcl::search::KdTree<pcl::PointXYZRGB>);

tree->setInputCloud (cloud); //KdTree 생성

std::vector<pcl::PointIndices> cluster\_indices; // 군집화된 결과물의 Index 저장, 다중 군집화 객체는 cluster\_indices[0] 순으로 저장

// 군집화 오브젝트 생성

pcl::EuclideanClusterExtraction<pcl::PointXYZRGB> ec;

ec.setInputCloud (cloud); // 입력

ec.setClusterTolerance (0.02); // 2cm

ec.setMinClusterSize (100); // 최소 포인트 수

ec.setMaxClusterSize (25000); // 최대 포인트 수

ec.setSearchMethod (tree); // 위에서 정의한 탐색 방법 지정

ec.extract (cluster\_indices); // 군집화 적용

// 클러스터별 정보 수집, 출력, 저장

int j = 0;

for (std::vector<pcl::PointIndices>::const\_iterator it = cluster\_indices.begin (); it != cluster\_indices.end (); ++it)

{

pcl::PointCloud<pcl::PointXYZRGB>::Ptr cloud\_cluster (new pcl::PointCloud<pcl::PointXYZRGB>);

for (std::vector<int>::const\_iterator pit = it->indices.begin (); pit != it->indices.end (); ++pit)

cloud\_cluster->points.push\_back (cloud->points[\*pit]);

cloud\_cluster->width = cloud\_cluster->points.size ();

cloud\_cluster->height = 1;

cloud\_cluster->is\_dense = true;

// 포인트수 출력

std::cout << "PointCloud representing the Cluster: " << cloud\_cluster->points.size () << " data points." << std::endl;

// 클러스터별 이름 생성 및 저장

std::stringstream ss;

ss << "cloud\_cluster\_" << j << ".pcd";

pcl::PCDWriter writer;

writer.write<pcl::PointXYZRGB> (ss.str (), \*cloud\_cluster, false); //\*

j++;

}

return (0);

}

실행 & 결과

$ PointCloud before filtering has: 23330 data points.

$ PointCloud representing the Cluster: 5981 data points.

$ PointCloud representing the Cluster: 5111 data points.

$ PointCloud representing the Cluster: 4431 data points.

$ PointCloud representing the Cluster: 2768 data points.

$ PointCloud representing the Cluster: 2513 data points.

$ PointCloud representing the Cluster: 1552 data points.

$ PointCloud representing the Cluster: 934 data points.

시각화 & 결과

$ pcl\_viewer cloud\_cluster\_0.pcd

$ pcl\_viewer cloud\_cluster\_6.pcd

참고: <https://pcl.gitbook.io/tutorial/part-2/part02-chapter01/part02-chapter01-euclidean-pcl-cpp>

<https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html>