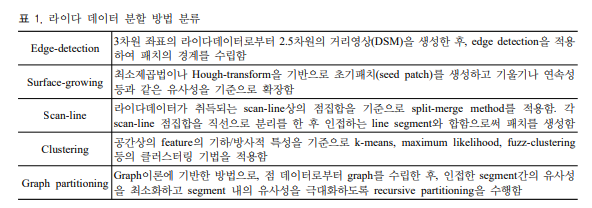
**사전조사**

2020.08.06

**클러스터링 군집화 개요(수고 측정 후보 1)**



출처: 김성준, 이임평 (2010). 라이다데이터 분할 알고리즘의 시뮬레이션 기반 성능평가. 대한공간정보학회 지, 18(2), 119-129

클러스터링(군집화)는 개체들이 주어졌을 때, 개체들을 몇 개의 클러스터(부분 그룹)으로 나누는 과정 의미. Unsupervised learning(비지도 학습)의 대표적인 예이다.

개체들을 그룹으로 나누는 과정을 통해 클러스터 내부 멤버들 사이는 서로 가깝거나 비슷하게, 서로 다른 두 클러스터 사이의 멤버 간에는 서로 멀거나 비슷하지 않게 하는 것이 클러스터링의 목표

왜 수고 측정의 방법 후보로 설정했는가?

->나무의 3차원 포인트 클라우드를 구성 했을 때 겹치는 부분이 없으면 각 개체목의 최대 높이를 구하면 되지만 나무의 모양은 제각각이므로 겹치는 부분이 생긴다면 각 개체목의 최대 높이를 구별하기 힘들다.

**즉, 포인트 클라우드 데이터 특정상 대상에 대한 참값을 알기 어렵기 때문에 분할을 할 수 있는 알고리즘이 필요한데 위 표에서 설명해준 분할의 방법 중 첫번째 후보로 클러스터링을 선정.**

위와 같은 이유로 각 개체목의 구성 포인트들을 군집화를 한 뒤 최대한 개체목을 구분을 한 후 높이를 구하는 방식으로 해결해 보려는 시도를 계획.

개체목의 좌표는 3차원 좌표이지만 군집화 개념을 설명 하기 위해 2차원 좌표로 개념 설명 및 이해.

만약 개체목들을 거리공간 안에 나타낼 수 있다면 개체와 개체 사이의 거리를 정의 가능하며 예를들어 유클리드 공간안에 개체목을 표현 가능하다면 유클리드 거리를 정의 가능. 이러한 경우 클러스터링의 목표는 같은 클러스터 내의 두 멤버들 사이의 거리를 최소화하고 서로 다른 두 클러스터 사이의 멤버 간의 거리를 최대화.

유클리드 거리의 필요충분조건은 아래와 같다.

거리 공간은 공간을 나타내는 집합 X와 거리 함수 d:X×X→[0,∞)로 이루어져서, 두 점 사이의 거리가 정의된 공간을 의미합니다. d가 거리 함수가 되려면, 아래와 같은 조건들을 만족해야 합니다.

임의의 x,y∈X에 대하여, d(x,y)=0과 x=y는 동치여야 합니다.

임의의 x,y∈X에 대하여, d(x,y)=d(y,x)를 만족하여야 합니다.

임의의 x,y,z∈X에 대하여, d(x,y)+d(y,z)≥d(z,x)를 만족하여야 합니다. 이 부등식은 주로 삼각부등식으로 잘 알려져 있습니다.

점인 경우에는 유클리드 거리, Lp norm 등의 거리를 정의할 수 있고, 집합의 경우에는 [자카드 지수](https://en.wikipedia.org/wiki/Jaccard_index) (교집합의 크기 / 합집합의 크기)를, 그리고 문자열의 경우에는 [편집 거리](https://en.wikipedia.org/wiki/Edit_distance)나 LCS(최대 부분 공통 문자열)의 길이를 이용하여 거리를 표현할 수 있습니다. <출처: 삼성 소프트웨어 멤버십 클러스터링(군집화) 개요>

\*유클리드 거리

유클리드 거리(Euclidean distance)는 [두 점 사이의 거리](https://ko.wikipedia.org/wiki/%EB%91%90_%EC%A0%90_%EC%82%AC%EC%9D%B4%EC%9D%98_%EA%B1%B0%EB%A6%AC)를 계산할 때 흔히 쓰는 방법이다. 이 거리를 사용하여 [유클리드 공간](https://ko.wikipedia.org/wiki/%EC%9C%A0%ED%81%B4%EB%A6%AC%EB%93%9C_%EA%B3%B5%EA%B0%84)을 정의할 수 있으며, 이 거리에 대응하는 [노름](https://ko.wikipedia.org/wiki/%EB%85%B8%EB%A6%84)을 유클리드 노름(Euclidean norm)이라고 부른다.

[직교 좌표계](https://ko.wikipedia.org/wiki/%EC%A7%81%EA%B5%90_%EC%A2%8C%ED%91%9C%EA%B3%84)로 나타낸 점 p = (p1, p2,..., pn)와 q = (q1, q2,..., qn)가 있을때, 두 유클리드 노름을 이용하여 두 점 p, q의 거리를 계산하면 다음과 같다.

<출처: 나무 위키>

‖p−q‖=(p−q)⋅(p−q)=‖p‖2+‖q‖2−2p⋅q.

\*유클리드 노름

n차원 유클리드 공간(=좌표평면)에서의 노름은 유클리드 노름(Euclidean norm)이라고 부르고 다음과 같이 정의한다.

∥x∥2=x12+x22+⋯+xn2=∑k=1nxk2∥x∥2​=x1​2+x2​2+⋯+xn​2​=k=1∑n​xk​2​

lp 노름에서 p=2의 꼴이다.  
  
어디서 많이 본 듯한 모양이라면 정답이다. 평면좌표(n=2)나 공간좌표(n=3)에서 원점과 점 x=(x1,x2,⋯ ,xn)x=(x1​,x2​,⋯,xn​) 사이의 거리이다. 이 노름에서 유도되는 거리 함수가 유클리드 공간에서 일반적으로 정해지는 거리 함수이며, 유클리드 거리 함수(Euclidean metric)라고 칭한다. 옛 문헌에는 피타고라스 거리 함수(Pythagorean metric)라고 표기된 문헌도 존재하지만 지금은 사장된 표현.

<출처: 나무 위키>

조사한 클러스터링 방식은 세 가지가 있다. Hirearchical Clustering과 Point Assignment Clustering, Density-based spatial clustering이다.

**Hirearchical Clustering** 같은 경우 개별 대상 간의 거리에 의하여 가장 가까이 있는 대상들로부터 시작하여 결합해 가는 방식.

즉, 탐욕적방식을 이용해 군집을 최소 단위로 쪼갠 후에 거리기반으로 다시 합쳐주면서 군집이 형성되는 과정을 정확하게 파악 할 수 있지만 자료의 크기가 크면 분석하기 어렵다는 단점이 있다. agglomerative clustering(병합 군집) 등의 방식의 예가 있다.

**Point Assignment Clustering**의 경우 군집의 수를 정한 상태에서 설정된 군집의 중심에서 가장 가까운 개체를 하나씩 포함하면서 빠르게 군집을 형성 할 수 있는 방식으로 군집의 수를 미리 정하고 초기값을 미리 정해주면서 구하는 방식이다. 하지만 미리 적절한 k개의 군집의 수를 정해야 하는 단점이 존재한다. 대표적으로 k-means 알고리즘이 있다.

**Density-based spatial clustering**의 경우 점이 몰려 있는 부분, 즉 밀도가 높은 부분을 클러스터링 하는 방식으로 대표적으로 DBSCAN 방식이 있다.

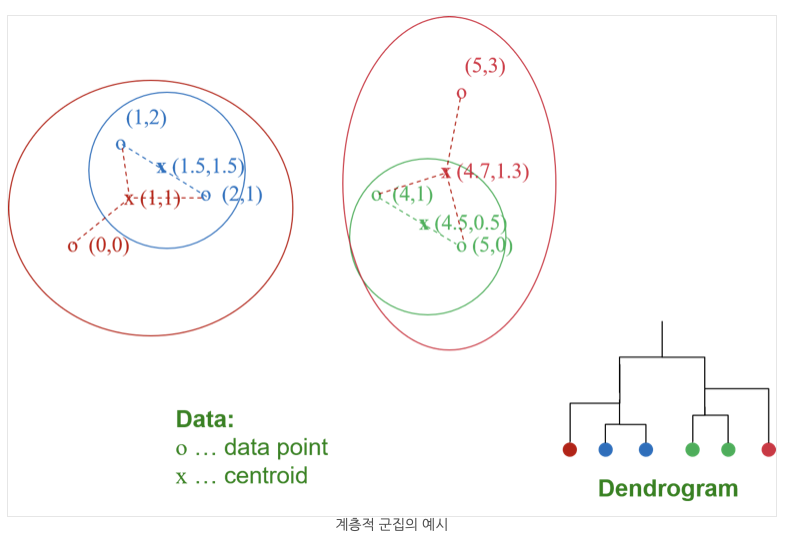
더 자세하게 각각의 방식에 대해서 알아보자면

**Hierarchical Clustering(계층적)**

**<병합군집법>**

병합군집은 반복적으로 두개의 가까운 클러스터를 찾는 것이다.

거리 공간에 두 점 사이의 거리를 구하기 위해서 거리를 구할 두 점을 구한다. 클러스터 내부에 있는 점의 좌표값의 평균을 이용하는 방법으로 Centroid의 좌표를 구한다. 이 방식으로 하나의 점이 추가 되면 다시 3개의 점의 평균으로 Centroid를 정해 주면서 군집화를 진행한다.



<출처: <https://www.secmem.org/blog/2019/05/17/clustering/>>

위와 같은 방식으로 두개의 가까운 클러스터를 반복적으로 찾는 것이다.

처음에는 (1,2)와 (2,1)의 원소를 갖는 파란색 원의 클러스터를 만들고 centroid를 (1.5,1.5)로 정한다.

(4,1)과 (5,0)을 원소로 갖는 녹색원의 클러스터을 만들고, centroid를 (4.5, 0.5)로 정한다.

그 다음은 가장 가까운 두개의 클러스터인 0,0을 원소로 가지는 클러스터, 파란색 원 모양의 클러스터를 합쳐서 왼쪽 빨간색 클러스터를 만들어 낸다.

마찬가지로 오른쪽 빨간색 클러스터 또한 같은 방식으로 만들어 진다.

시간 복잡도는 집합의 원소 N개라 가정할 때 매번 두 클러스터 사이의 거리를 모두 계산하고 합칠 때 마다 O(n^2)의 시간이 소요 되고 총 합치는 횟수가 N-1번 이므로 O(N^3)이 걸려 매우 큰 복잡도를 가진다.

우선순위 큐를 사용한다면 O(N^2log N)이 걸린다.

**Point Assignment Clustering(비계층적)**

**<K-means>**

각 클러스터가 어떤 개체를 가지는 지 기록하는 방식으로 진행되는 클러스터링 방법.

대표적으로 K-means알고리즘이 존재.

K-means알고리즘은 임의로 선택한 K개의 점을 이용해 초기의 클러스터 K개를 만들고 클러스터를 계속 맞춰서 변화하며 군집화를 시키는 방식으로써

임의의 k개의 점은 각 나무의 개수만큼 해당 나무의 중심점이나 기타 다른 방법으로 설정 후에 구해준다.

선택한 k개의 점을 centorid로 설정을 한 후 초기 클러스터 k개를 만든다.

각 클러스터 마다 centroid를 새로 찾으며 다시 각 점마다 가장 가까운 centroid를 찾아서 클러스터를 만들고 다시 centroid를 조정하는 작업을 반복한다.

클러스터링한 결과가 달라지지 않을 때까지 반복.

시계, 그리기이(가) 표시된 사진

자동 생성된 설명

시계, 그리기이(가) 표시된 사진

자동 생성된 설명

<출처: <https://www.secmem.org/blog/2019/05/17/clustering/>>

위와 같은 방식으로 점 여덟개에서 K-means 클러스터링을 진행하면 K = 2로 설정 했을 때 centroid를 주황색점과 파란점으로 설정을 하고 그 centroid를 기반으로 가장 가까운 점을 찾고 클러스터를 만든다. 그 후 다시 새로운 centroid인 네모점을 구해준다.

이런 방식으로 계속해서 centroid를 설정을 해주고 가장 가까운 점을 기준으로 클러스터링을 시켜줌 으로써 사진의 가장 마지막과 같이 더이상 클러스터의 변화가 없을 때 까지 군집화를 완료한다.

k-means 알고리즘은 처음에 k를 선택을 하고 클러스터링을 진행하는데 k를 작게 한다면 다른 클러스터에 위치해야 하는 점이 한 클러스터에 묶이게 되는 문제가 발생 하고 품질이 하락함.

또한 K를 너무 크게 한다면 거리가 충분히 가까워서 한클러스터에 묶일 수 있는 부분이 다른 클러스터에 묶일 가능 성이 있다. 수고의 경우 개체목의 개수를 k로 설정하고 그만큼 군집화를 해야하는데 이런 경우 생기는 문제점에 대해 테스트 할 필요성이 있다.

참고로 최적의 k를 구하는 방법에는 k를 1로 놓고 시작하여 2배씩 늘리면서 평균거리가 적게 줄어드는 k = 2^i ~ 2^i+1사이의 i를 찾아 2^i-1 ~ 2^i 사이의 k값이 최적이 k값이다. 좀 더 정확한 값을 얻으려면 2^i-1 ~ 2^i사이에 이분탐색을 사용하여 구한다. 예를 들어, 현재 최적의 k가 x 이상 y 이하의 닫힌 구간에 존재한다는 사실을 안다면, 먼저 z=(x+y)/2로 놓고, k=z로 놓고 k-means 클러스터링을 진행한다. 이 때, k=z와 k=y 사이에서 평균 거리가 적게 줄어들었다면, 최적의 k는 닫힌 구간 [x,z]에 존재한다고 생각할 수 있다. 반대의 경우에는 최적의 k가 닫힌 구간 [z,y]에 존재한다고 생각할 수 있다.

시간 복잡도는 k 점을 찾는 과정에서 O(k) 그 후 각 점에서 가까운 centroid를 찾는 과정이 O(Nk), centroid를 조정 할 때 O(N), 이러한 과정을 x번 반복시 O(Nkx)

만일 최적의 k를 찾는 방법을 적용한다면 O(NkxlogN)이 된다. (이분탐색의 시간 복잡도: O(log N))

최적의 클러스터를 형성할 때 제대로 클러스터가 형성 되지 않는 문제점이 발생할 수 있으므로 추후 이에 대해 논의 해봐야하며 위에서 설명한 un-supervised 방식과는 다른 종류의 supervised 러닝인 KNN알고리즘에 대해서 논의 해봐야 한다.

\*K-means 사용 예

<https://eunsukimme.github.io/ml/2019/12/16/K-Means/>

**Density-based spatial clustering(밀도 기반)**

**<DBSCAN>**

클러스터를 데이터가 높은 밀도로 모여 있는 공간으로 보는 기법

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

**장점**

1. K-Means와 같이 클러스터의 수를 정하지 않아도 된다.

2. 클러스터의 밀도에 따라서 클러스터를 서로 연결하기 때문에 기하학적인 모양을 갖는 군집도 잘 찾을 수 있다.

**단점**

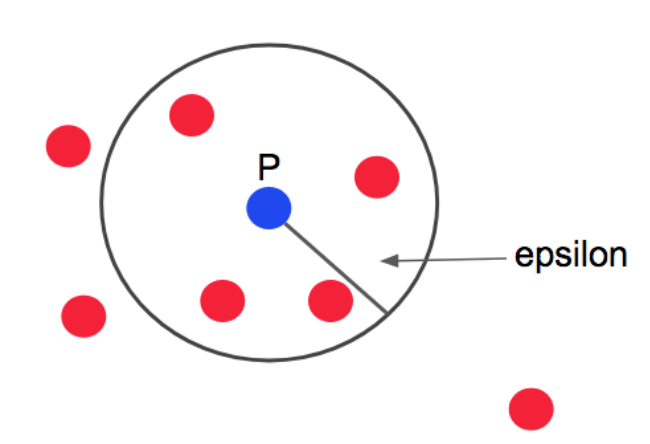
1. 많은 연산을 수행하기에 K 평균에 비해 그 속도가 느림
2. 반지름과 임계치 설정에 많은 영향을 받는다
3. 유클리드 제곱 거리를 사용하는 모든 데이터 모델의 공통적인 단점인, 'Curse Of dimensionality 또한 존재

* 이는 2차원이나 3차원 등 차원 수가 낮은 데이터셋에는 문제가 되지 않지만, 고차원 데이터세트로 갈수록 필요한 학습 데이터 양이 급증하는 문제점이며, 이 때문에 많은 연산이 필요해진다는 단점이 있다.

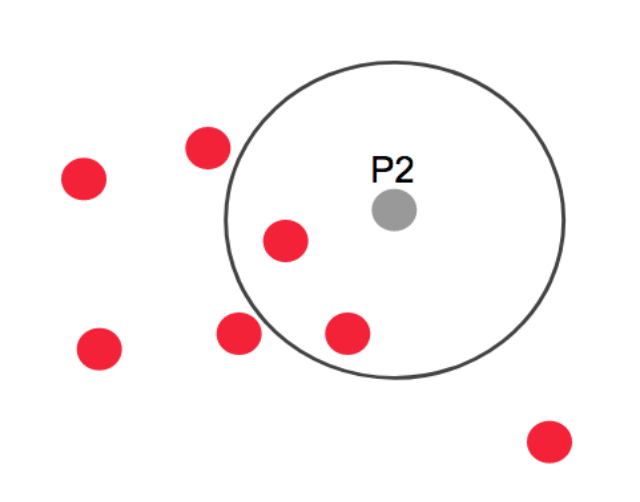
먼저 점 p가 있다고 할때, 점 p에서 부터 거리 e (epsilon)내에 점이 m(minPts) 개 있으면 하나의 군집으로 인식한다고 하자. 이 조건 즉 거리 e 내에 점 m개를 가지고 있는 점 p를 core point(중심점) 이라고 한다.

DBSCAN 알고리즘을 사용하려면 기준점 부터의 거리 epsilon값과, 이 반경내에 있는 점의 수 minPts를 인자로 전달해야 한다.

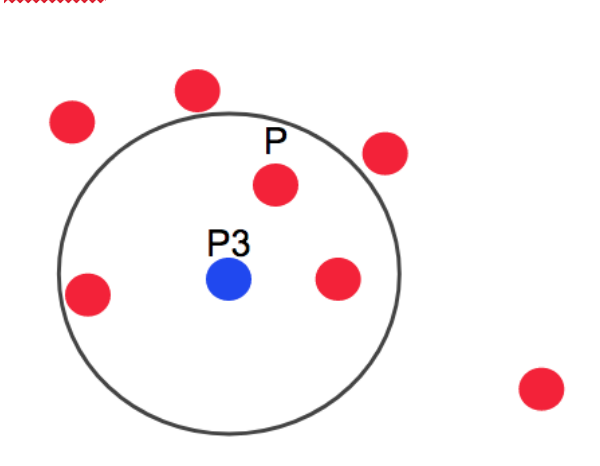
아래 그림에서 minPts = 4 라고 하면, 파란점 P를 중심으로 반경 epsilon 내에 점이 4개 이상 있으면 하나의 군집으로 판단할 수 있는데, 아래 그림은 점이 5개가 있기 때문에 하나의 군집으로 판단이 되고, P는 core point가 된다.



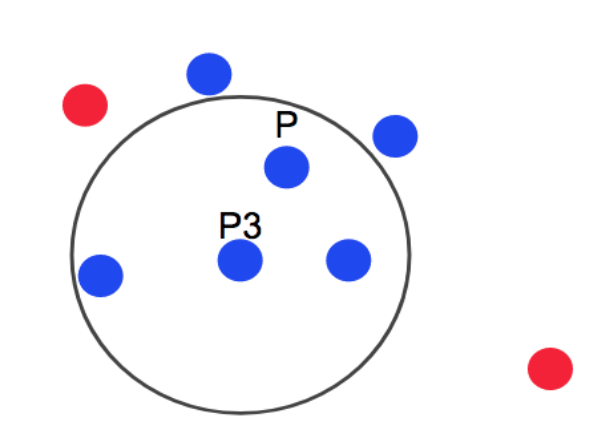
아래 그림에서 회색점 P2의 경우 점 P2를 기반으로 epsilon 반경내의 점이 3개 이기 때문에, minPts=4에 미치지 못하기 때문에, 군집의 중심이 되는 core point는 되지 못하지만, 앞의 점 P를 core point로 하는 군집에는 속하기 때문에 이를 boder point (경계점)이라고 한다.



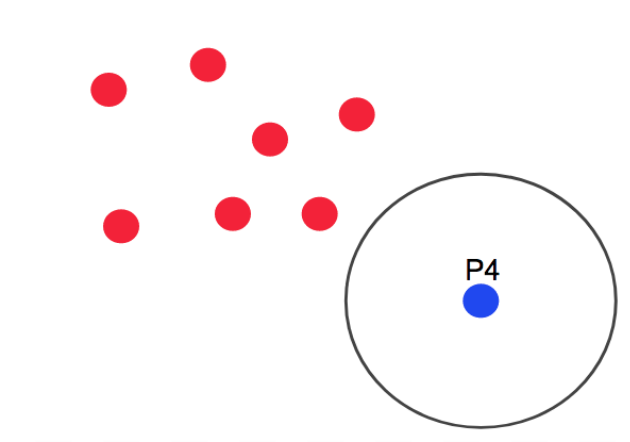
아래 그림에서 P3는 epsilon 반경내에 점 4개를 가지고 있기 때문에 core point가 된다.



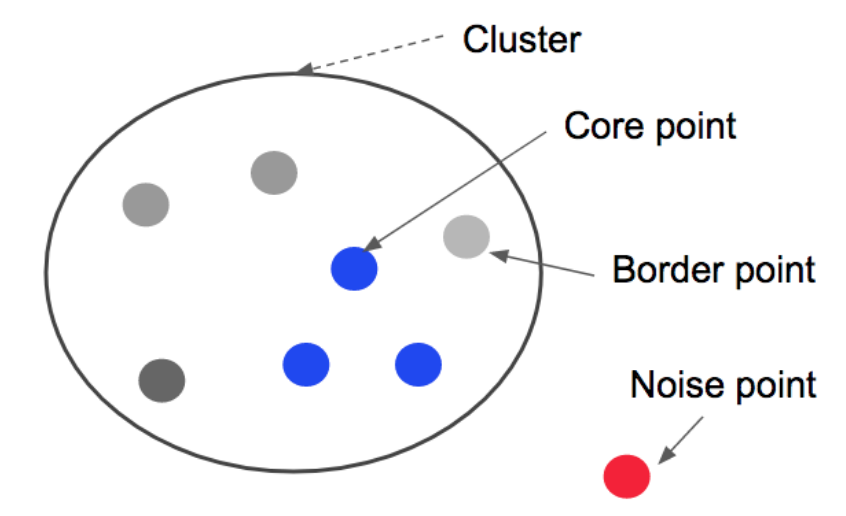
그런데 P3를 중심으로 하는 반경내에 다른 core point P가 포함이 되어 있는데, 이 경우 core point P와  P3는 연결되어 있다고 하고 하나의 군집으로 묶이게 된다.



마지막으로 아래 그림의 P4는 어떤 점을 중심으로 하더라도 minPts=4를 만족하는 범위에 포함이 되지 않는다. 즉 어느 군집에도 속하지 않는 outlier가 되는데, 이를 noise point라고 한다.



이를 모두 정리해보면 다음과 같은 그림이 나온다.



1. 점을 중심으로 epsilon 반경내에 minPts 이상 수의 점이 있으면 그 점을 중심으로 군집이 형성되고 그 점이 core point 이다. Core point 가 서로 다른 core point의 군집의 일부가 되면 그 군집을 서로 연결되어 있다고 하고 하나의 군집으로 연결을 한다.
2. 군집에는 속하지만, 스스로 core point가 안되는 점을 border point라고 하고, 주로 클러스터의 외곽을 이루는 점이 된다.
3. 그리고 어느 클러스터에도 속하지 않는 점은 Noise point가 된다.

|  |  |  |
| --- | --- | --- |
|  |  |  |

**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)

**참고**

<https://www.secmem.org/blog/2019/05/17/clustering/>

[J. Leskovec, A. Rajaraman, J. Ullman: Mining of Massive Datasets, Chapter. 7. Clustering](J.%20Leskovec,%20A.%20Rajaraman,%20J.%20Ullman:%20Mining%20of%20Massive%20Datasets,%20Chapter.%20%20%207.%20Clustering) (<http://www.mmds.org>)

<https://bcho.tistory.com/1205> [조대협의 블로그]

**환경 구성**

환경을 구성하는데 pcl 자체에 의존성 패키지들이 많고 버전이 다양하여 사양 부족, 환경 변수 블일치등 설치에 난항을 겪음. C++환경 설치와 파이썬 환경 설치 및 구동 방식 또한 다름.

* 여러 설치법을 찾아 본 후 리눅스 우분투 16.04LTS 버전으로 변경하고 오류들을 해결해 나가면서 구축 완료
* 그러나 현재 c++, python 기반 라이브러리들이 동작은 하나 c++ 기반의 경우 고연산 작업에 들어가면 계속하여 오류 발생. 해결방안 찾는 중.

깃허브(<https://github.com/gofmsldks/2020_FosCo>)에 Readme에 해결 방법들이 나와있는 사이트들을 첨부.

**다음은 해결 방법에 대한 일부분**

산림대융합프로젝트 과제명: 산림조사 효율성 제고를 위한 3d레이저 스캐너의 pcd 처리 시스템 개발

@pcl 설치법 -><https://readthedocs.org/projects/python-pcl-fork/downloads/pdf/rc_patches4/>

@pcl docker -><https://hub.docker.com/r/youyue/pcl-docker/>

@우분투 설치법 c++ 버전 pcl설치법 ->

1. <https://ko.a-ubuntu.com/q/c-%EC%9A%A9-ubuntu-16042-lts%EC%97%90-point-cloud-library-v18-pcl-180%EC%9D%84-%EC%84%A4%EC%B9%98%ED%95%98%EB%8A%94-%EB%B0%A9%EB%B2%95%EC%9D%80-%EB%AC%B4%EC%97%87%EC%9E%85%EB%8B%88%EA%B9%8C-%EB%8B%AB%EC%9D%80-19787> <https://gist.github.com/IgniparousTempest/ce5fadbe742526d10d6bdbf15c3a3fe7>
2. 오류해결법 :<http://jinyongjeong.github.io/2017/01/09/pcl_install_with_qt5/>

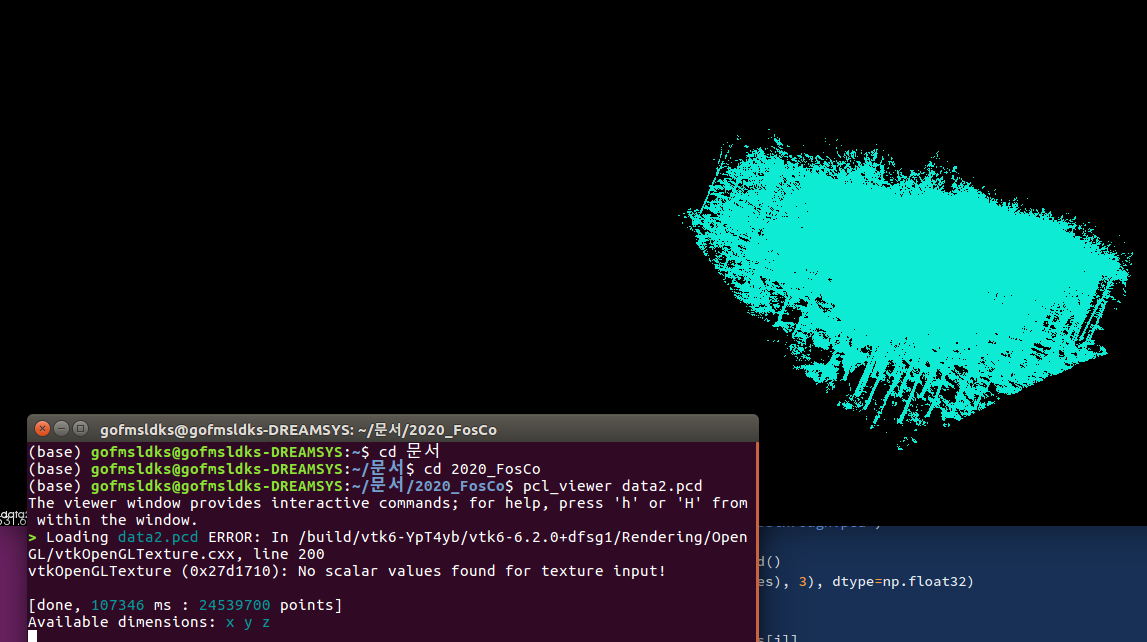
@pcl 전반적인 설명(c++, python 설치방법도 나와있음, 제일 깔끔) -><https://adioshun.gitbooks.io/pcl/content/pytncloud.html>

@아나콘다 파이썬 설치시 Can not import pcl, boost version error. #285 오류 해결법 -><https://github.com/strawlab/python-pcl/issues/285>

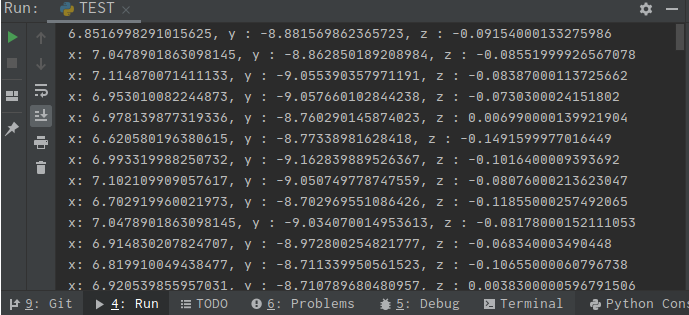
@pts 파일을 pcd로 바꿔주는 코드 -><https://gist.github.com/omair18/f3b85a7085b5b006c5ba919269f1db3c>

환경 구축 후 제공 받은 data.pts파일을 이용하여 pcl\_viewer, pts파일을 pcd로 변환, 각 포인트들을 출력해 보았고 기본적인 부분들은 정상적으로 구동 됨을 확인.









**20.08.06 이후 단기 계획**

1. 각 클러스터링 방식의 라이브러리나 예시 코드들 분석.
2. 이번 프로젝트의 전반적인 기본 코드를 분석해보고 각 함수 부분 도식화, 필요한 부분 수정 및 설계
3. 분석한 클러스터링을 적용한 수고 추출 알고리즘 도입
4. 테스트 후 적절한 방식을 찾을 때 까지 반복
5. 속도 문제 발생시 파이썬 내부에 c++을 연동하는 방안 연구

**\*질문사항**

위에서 조사해온 기본적인 클러스터링 방식외에 더 좋은 알고리즘이 있는지?

추가적으로 제가 놓친 점이 있는지?

떠오르는 다른 방식의 방법이 있으신지?