K-means 알고리즘 구현

2020.08.10 ~

1. 사용한 모듈

```
import pandas as pd
from sklearn import datasets
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
import numpy as np
from mpl_toolkits.mplot3d import Axes3D
import pcl
```

KMeans 모듈 설명 및 사용법

```
  class KMeans(TransformerMixin, ClusterMixin, BaseEstimator):
 """K-Means clustering.
    Read more in the :ref:`User Guide <k_means>`.
    Parameters
    ----------
    n_clusters : int, default=8
        The number of clusters to form as well as the number of
        centroids to generate.
    init : {'k-means++', 'random', ndarray, callable}, default='k-means++'
        Method for initialization:
        'k-means++' : selects initial cluster centers for k-mean
        clustering in a smart way to speed up convergence. See section
        Notes in k_init for more details.
        'random': choose `n_clusters` observations (rows) at random from data
        for the initial centroids.
        If an ndarray is passed, it should be of shape (n_clusters, n_features)
        and gives the initial centers.
        If a callable is passed, it should take arguments X, n_clusters and a
        random state and return an initialization.
    n_init : int, default=10
        Number of time the k-means algorithm will be run with different
        centroid seeds. The final results will be the best output of
        n_init consecutive runs in terms of inertia.
    max_iter : int, default=300
        Maximum number of iterations of the k-means algorithm for a
        single run.
    tol : float, default=1e-4
        Relative tolerance with regards to Frobenius norm of the difference
        in the cluster centers of two consecutive iterations to declare
        convergence.
        It's not advised to set `tol=0` since convergence might never be
        declared due to rounding errors. Use a very small number instead.
    precompute_distances : {'auto', True, False}, default='auto'
        Precompute distances (faster but takes more memory).
        'auto' : do not precompute distances if n_samples * n_clusters > 12
        million. This corresponds to about 100MB overhead per job using
        double precision.
        True : always precompute distances.
        False : never precompute distances.
        .. deprecated:: 0.23
            'precompute_distances' was deprecated in version 0.22 and will be
            removed in 0.25. It has no effect.
    verbose : int, default=0
```

```
        Verbosity mode.
    random_state : int, RandomState instance, default=None
        Determines random number generation for centroid initialization. Use
        an int to make the randomness deterministic.
        See :term:`Glossary <random_state>`.
    copy_x : bool, default=True
        When pre-computing distances it is more numerically accurate to center
        the data first. If copy_x is True (default), then the original data is
        not modified. If False, the original data is modified, and put back
        before the function returns, but small numerical differences may be
        introduced by subtracting and then adding the data mean. Note that if
        the original data is not C-contiguous, a copy will be made even if
        copy_x is False. If the original data is sparse, but not in CSR format,
        a copy will be made even if copy_x is False.
    n_jobs : int, default=None
        The number of OpenMP threads to use for the computation. Parallelism is
        sample-wise on the main cython loop which assigns each sample to its
        closest center.
        ``None`` or ``-1`` means using all processors.
        .. deprecated:: 0.23
            ``n_jobs`` was deprecated in version 0.23 and will be removed in
            0.25.
    algorithm : {"auto", "full", "elkan"}, default="auto"
        K-means algorithm to use. The classical EM-style algorithm is "full".
        The "elkan" variation is more efficient on data with well-defined
        clusters, by using the triangle inequality. However it's more memory
        intensive due to the allocation of an extra array of shape
        (n_samples, n_clusters).
        For now "auto" (kept for backward compatibiliy) chooses "elkan" but it
        might change in the future for a better heuristic.
        .. versionchanged:: 0.18
            Added Elkan algorithm
    Attributes
    ----------
    cluster_centers_ : ndarray of shape (n_clusters, n_features)
        Coordinates of cluster centers. If the algorithm stops before fully
        converging (see ``tol`` and ``max_iter``), these will not be
        consistent with ``labels_``.
    labels_ : ndarray of shape (n_samples,)
        Labels of each point
    inertia_ : float
        Sum of squared distances of samples to their closest cluster center.
    n_iter_ : int
        Number of iterations run.
    See also
    --------
    MiniBatchKMeans
        Alternative online implementation that does incremental updates
        of the centers positions using mini-batches.
        For large scale learning (say n_samples > 10k) MiniBatchKMeans is
        probably much faster than the default batch implementation.
    Notes
    -----
    The k-means problem is solved using either Lloyd's or Elkan's algorithm.
    The average complexity is given by O(k n T), were n is the number of
    samples and T is the number of iteration.
    The worst case complexity is given by O(n^(k+2/p)) with
    n = n_samples, p = n_features. (D. Arthur and S. Vassilvitskii,
    'How slow is the k-means method?' SoCG2006)
    In practice, the k-means algorithm is very fast (one of the fastest
    clustering algorithms available), but it falls in local minima. That's why
    it can be useful to restart it several times.
    If the algorithm stops before fully converging (because of ``tol`` or
    ``max_iter``), ``labels_`` and ``cluster_centers_`` will not be consistent,
    i.e. the ``cluster_centers_`` will not be the means of the points in each
    cluster. Also, the estimator will reassign ``labels_`` after the last
    iteration to make ``labels_`` consistent with ``predict`` on the training
    set.
    Examples
    --------
    >>> from sklearn.cluster import KMeans
    >>> import numpy as np
    >>> X = np.array([[1, 2], [1, 4], [1, 0],
    ...               [10, 2], [10, 4], [10, 0]])
    >>> kmeans = KMeans(n_clusters=2, random_state=0).fit(X)
    >>> kmeans.labels_
    array([1, 1, 1, 0, 0, 0], dtype=int32)
    >>> kmeans.predict([[0, 0], [12, 3]])
    array([1, 0], dtype=int32)
    >>> kmeans.cluster_centers_
    array([[10.,  2.],
           [ 1.,  2.]])
    """
```

2. 구현 과정

pcl.load 로 data.pcd 파일 로드 후 로드한 데이터를 쉽게 연산하기 위해 Numpy 형으로 변환 시켜준다.

```
pc = pcl.load("data2.pcd") # "pc.from_file" Deprecated

pc_array = pc.to_array() # pc to Numpy
```

Numpy 로 변환된 pc_array 를 pandas 의 DataFrame 에 넣어 준 후 X,Y,Z 의 colum 을 붙여준다.

```
data = pd.DataFrame(pc_array)
data.columns=['X','Y','Z']

print(data)


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

            X          Y             Z

0      -0.80956 -0.43622   9.68732

1       -0.01456 -0.91722   9.79032

2       -0.13956 -2.06122  10.09632

3       -0.17356 -0.20022  10.12432

4       -0.93556 -0.11922  10.69232
```

```
...           ...      ...       ...

10031664  0.08844  0.03178   0.20532

10031665  0.09444  0.03078   0.15432

10031666  0.08544  0.02178   0.24032

10031667  0.08944  0.05078  -0.04568

10031668  0.09344  0.05578  -0.11068



[10031669 rows x 3 columns]
```

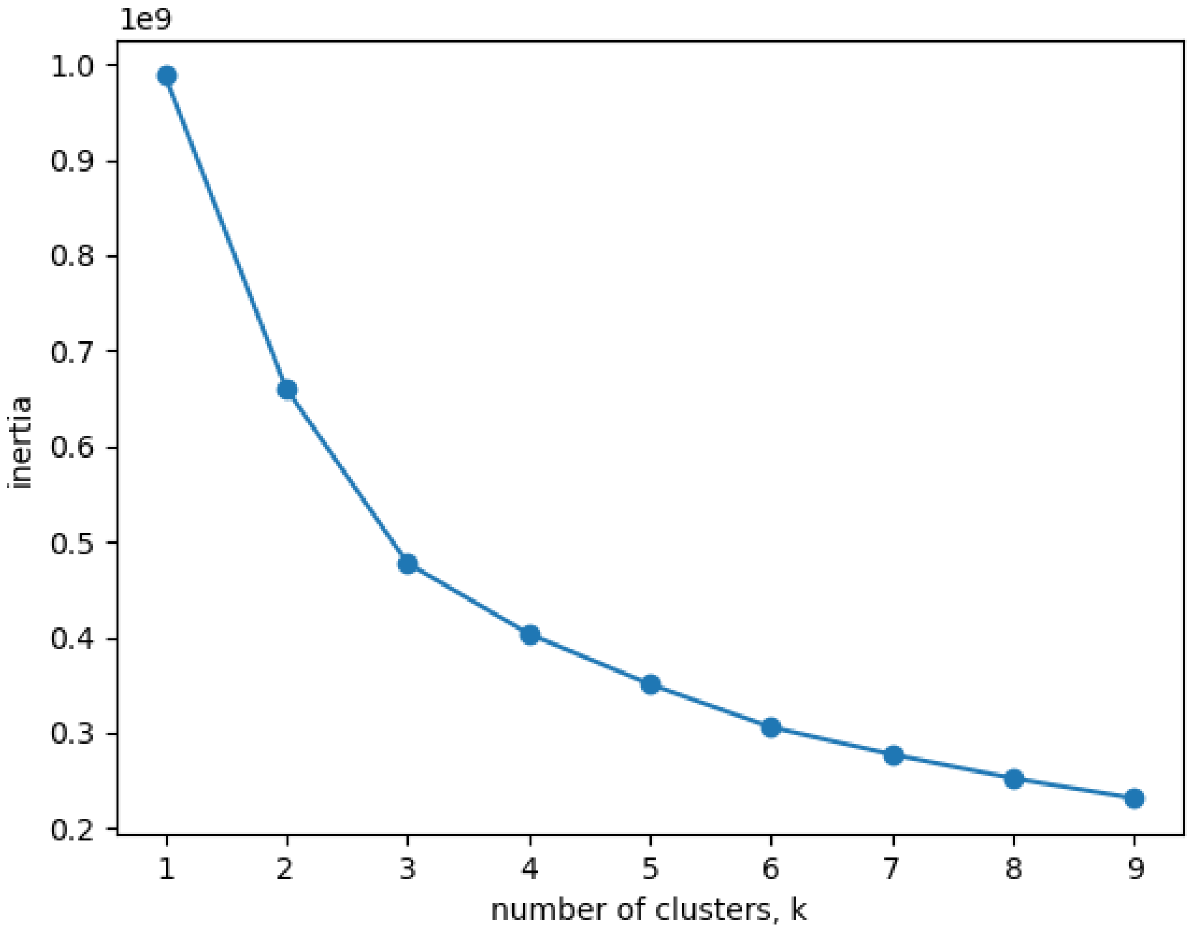
Inertia value 를 이용한 적정 군집수 판단

K Means 를 수행하기전에 k 개의 클러스터 개수를 정해줘야 한다. 그렇다면 몇 개의 클러스터의 수가 가장 적절할지 결정할 필요가 있다. 여기서 Inertia value 라는 값을 보면 적정 클러스터 수를 선택할 수 있는 기준을 얻을 수 있는데 Inertia value 는 군집화가된 후에, 각 중심점에서 군집의 데이타간의 거리를 합산을 하여 군집의 응집도를 나타내는 값이다, 이 값이 작을 수록 응집도가 높게 군집화가 잘되었다고 평가할 수 있다.

inertia value 는 KMeans 모델이 학습된 후에, model.inertia_ 값으로 볼 수 있다.

다음은 iris 데이타를 가지고 1~k 개의 클러스터로 클러스터링을 했을때, 각 클러스터 개수별로 inertia value 를 출력해 볼 수 있다.

```
# we can find appropriate the number of clusters with Inertia

ks = range(1, 5) # range denpend on k's number
inertias = []
for k in ks:
      model = KMeans(n_clusters=k)
      model.fit(data)
      inertias.append(model.inertia_)

# Plot ks vs inertias
plt.plot(ks, inertias, '-o')
plt.xlabel('number of clusters, k')
plt.ylabel('inertia')
plt.xticks(ks)
plt.show()
```



1e9
inertia
number of clusters, k

군집화 모델을 만들고 예측하는 부분

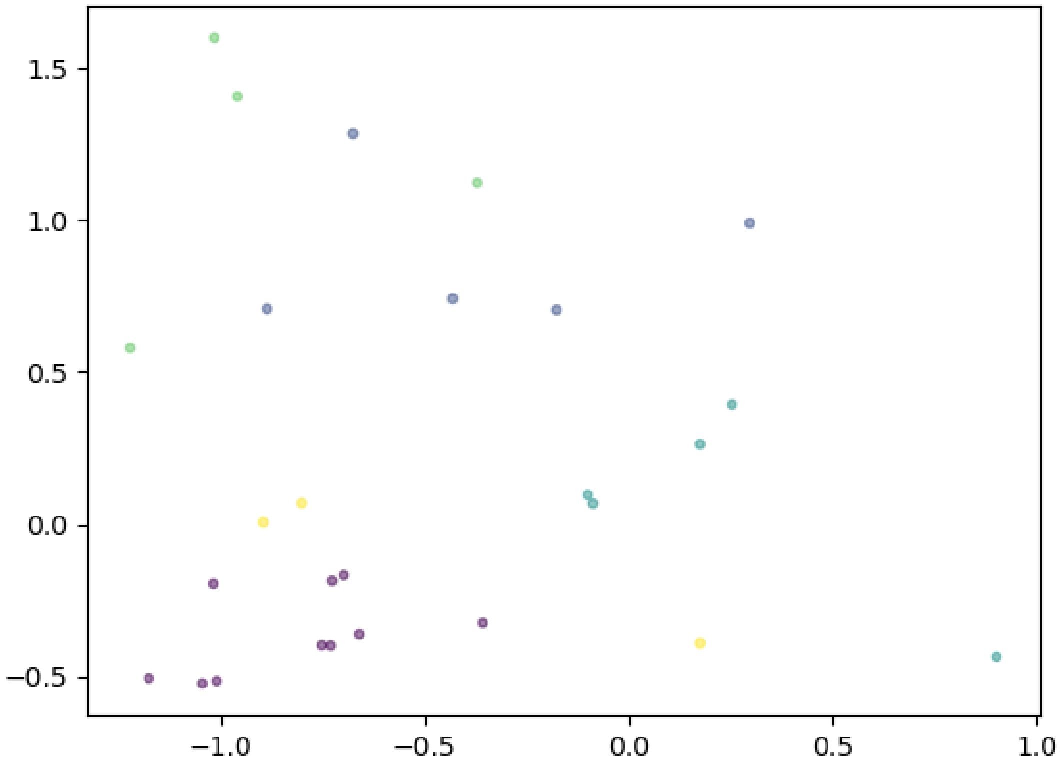
```
# create model and prediction

k_Num = int(input("input k: "))

model = Kmeans(n_clusters=k_Num ,algorithm='auto')
model.fit(data)
predict = pd.DataFrame(model.predict(data))
predict.columns=['predict']

# concatenate labels to df as a new column

r = pd.concat([data,predict],axis=1)

print(r)

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

input k: 20

               X           Y            Z        predict

0        -0.80956 -0.43622   9.68732        9

1        -0.01456 -0.91722   9.79032        9

2        -0.13956 -2.06122  10.09632        9

3        -0.17356 -0.20022  10.12432        9

4        -0.93556 -0.11922  10.69232        9

...           ...      ...       ...      ...

10031664  0.08844  0.03178   0.20532       10

10031665  0.09444  0.03078   0.15432       10

10031666  0.08544  0.02178   0.24032       10

10031667  0.08944  0.05078  -0.04568       10

10031668  0.09344  0.05578  -0.11068       10



[10031669 rows x 4 columns]
```

2D plot 과 3D plot 으로 PCD 데이터와 군집화 CENTROID 표시

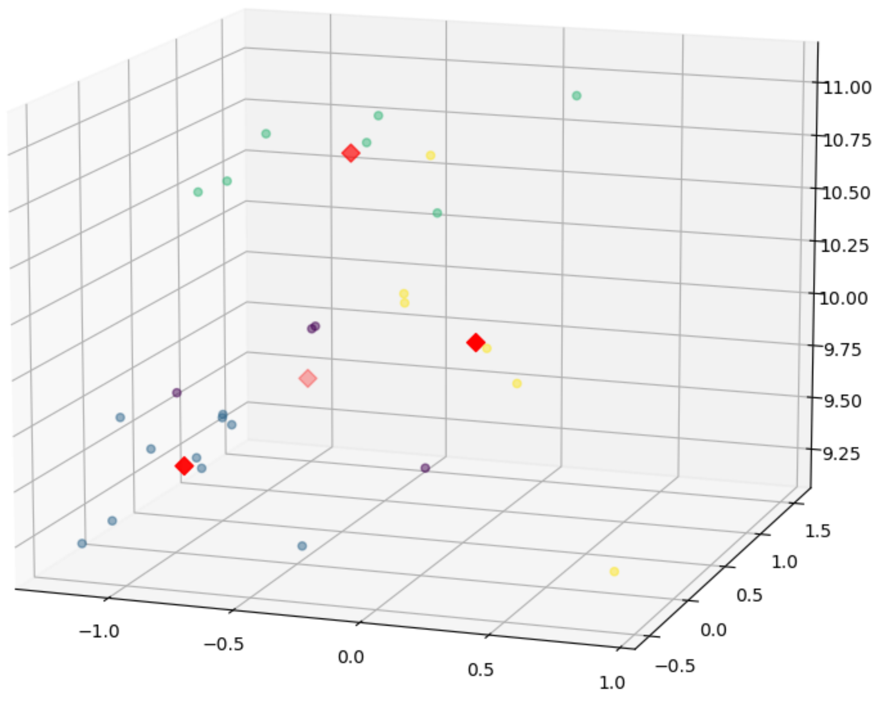
```python
# Clustering data visualization

#2D plot
plt.scatter(r['X'], r['Y'], r['Z'], c=r['predict'], alpha=0.5)
# 3d plot
fig = plt.figure(figsize=(10, 5))
ax = fig.add_subplot(111, projection='3d') # Axe3D object
ax.scatter(r['X'], r['Y'], r['Z'], c=r['predict'], alpha=0.5)
centers = pd.DataFrame(model.cluster_centers_,columns=['X','Y','Z'])
center_x = centers['X']
center_y = centers['Y']
center_z = centers['Z']
# if you want to see 2D centroid plot, delete below '#'
# plt.scatter(center_x,center_y,center_z, marker='D',c='b')
# if you want to see 3D centroid plot
ax.scatter(center_x,center_y,center_z,s=50, marker='D',c='r')
plt.show()
```

일부 test 데이터를 뽑아 적용 했을때

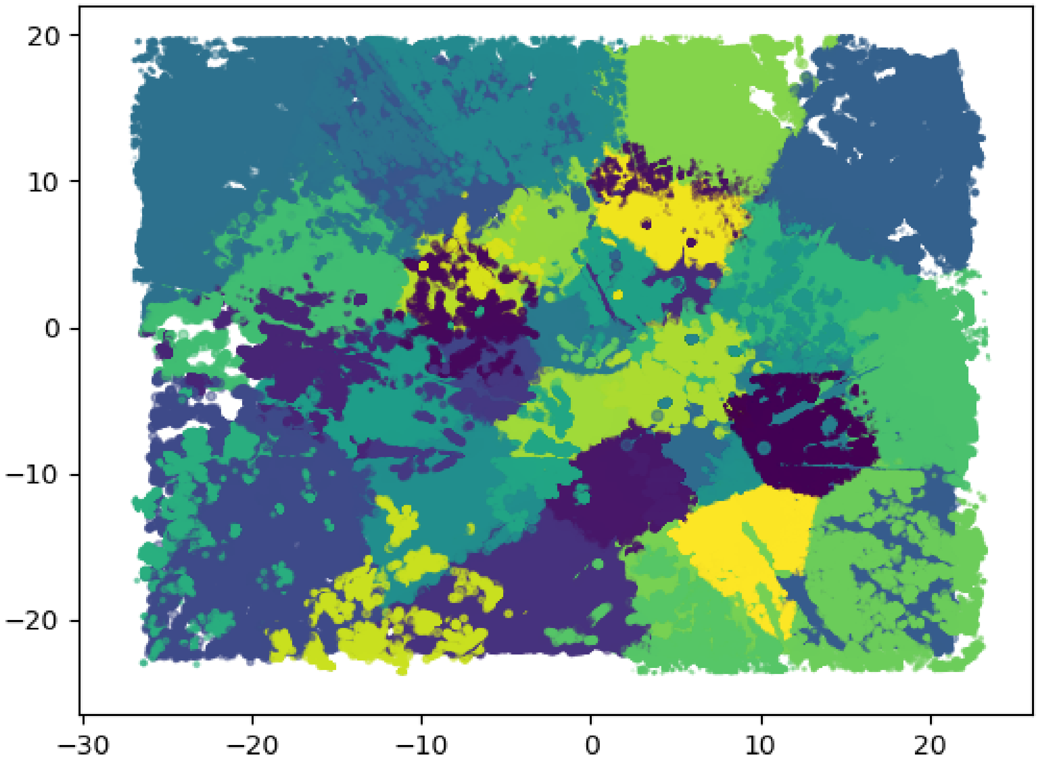
2d(k = 4 로 가정)



3d(k = 4 로 가정)



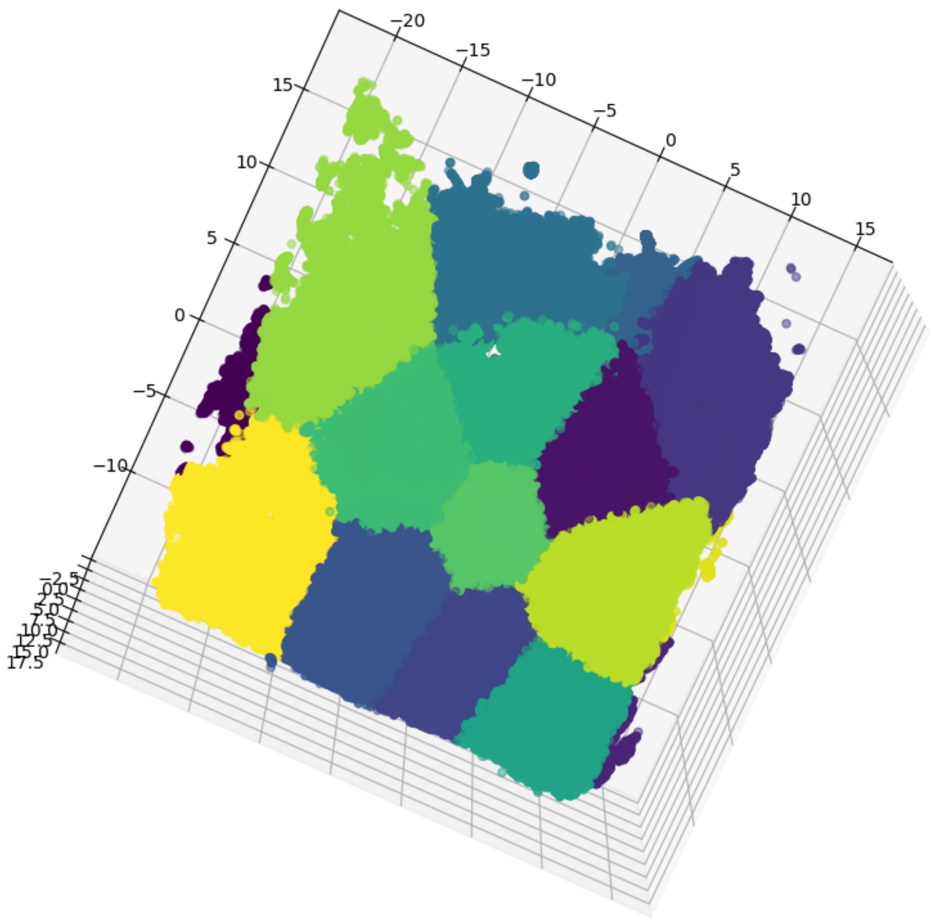
전체 데이터로 k 를 50 이라 가정하고 군집화를 했을때의 2D plot

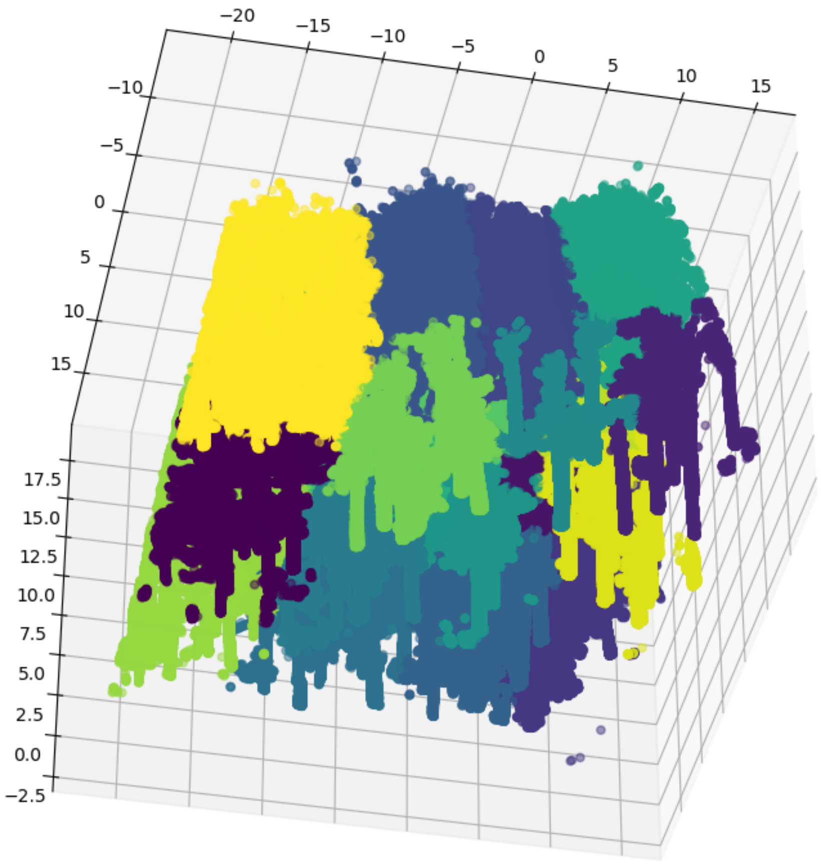


메모리 초과로 인한 3d plot 오류 – 2020.08.14

```
/home/gofmsldks/anaconda3/lib/python3.6/site-packages/matplotlib/collections.py:922:
RuntimeWarning: invalid value encountered in sqrt
  scale = np.sqrt(self._sizes) * dpi / 72.0 * self._factor
Traceback (most recent call last):
  File "/home/gofmsldks/anaconda3/lib/python3.6/site-
packages/matplotlib/backends/backend_qt5.py", line 480, in _draw_idle
    self.draw()
  File "/home/gofmsldks/anaconda3/lib/python3.6/site-
packages/matplotlib/backends/backend_agg.py", line 407, in draw
    self.figure.draw(self.renderer)




File "/home/gofmsldks/anaconda3/lib/python3.6/site-packages/matplotlib/artist.py", line 41, in
draw_wrapper
    return draw(artist, renderer, *args, **kwargs)
  File "/home/gofmsldks/anaconda3/lib/python3.6/site-packages/matplotlib/figure.py", line 1864, in
draw
    renderer, self, artists, self.suppressComposite)
  File "/home/gofmsldks/anaconda3/lib/python3.6/site-packages/matplotlib/image.py", line 132, in
_draw_list_compositing_images
    a.draw(renderer)
  File "/home/gofmsldks/anaconda3/lib/python3.6/site-packages/matplotlib/artist.py", line 41, in
draw_wrapper
    return draw(artist, renderer, *args, **kwargs)
  File "/home/gofmsldks/anaconda3/lib/python3.6/site-packages/mpl_toolkits/mplot3d/axes3d.py",
line 447, in draw
    reverse=True)):
  File "/home/gofmsldks/anaconda3/lib/python3.6/site-packages/mpl_toolkits/mplot3d/axes3d.py",
line 446, in <lambda>
    key=lambda col: col.do_3d_projection(renderer),
  File "/home/gofmsldks/anaconda3/lib/python3.6/site-packages/mpl_toolkits/mplot3d/art3d.py",
line 511, in do_3d_projection
    fcs = mcolors.to_rgba_array(fcs, self._alpha)
  File "/home/gofmsldks/anaconda3/lib/python3.6/site-packages/matplotlib/colors.py", line 301, in
to_rgba_array
    result = c.copy()
MemoryError: Unable to allocate 749. MiB for an array with shape (24539700, 4) and data type
float64

Process finished with exit code 130 (interrupted by signal 2: SIGINT)
```

2020.08.19 바닥데이터 제거 후 3d plot 을 구동 (k=20)이라 가정





−20
−15
−10
−5
0
5
10
15
−10
−5
0
5
10
15
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5

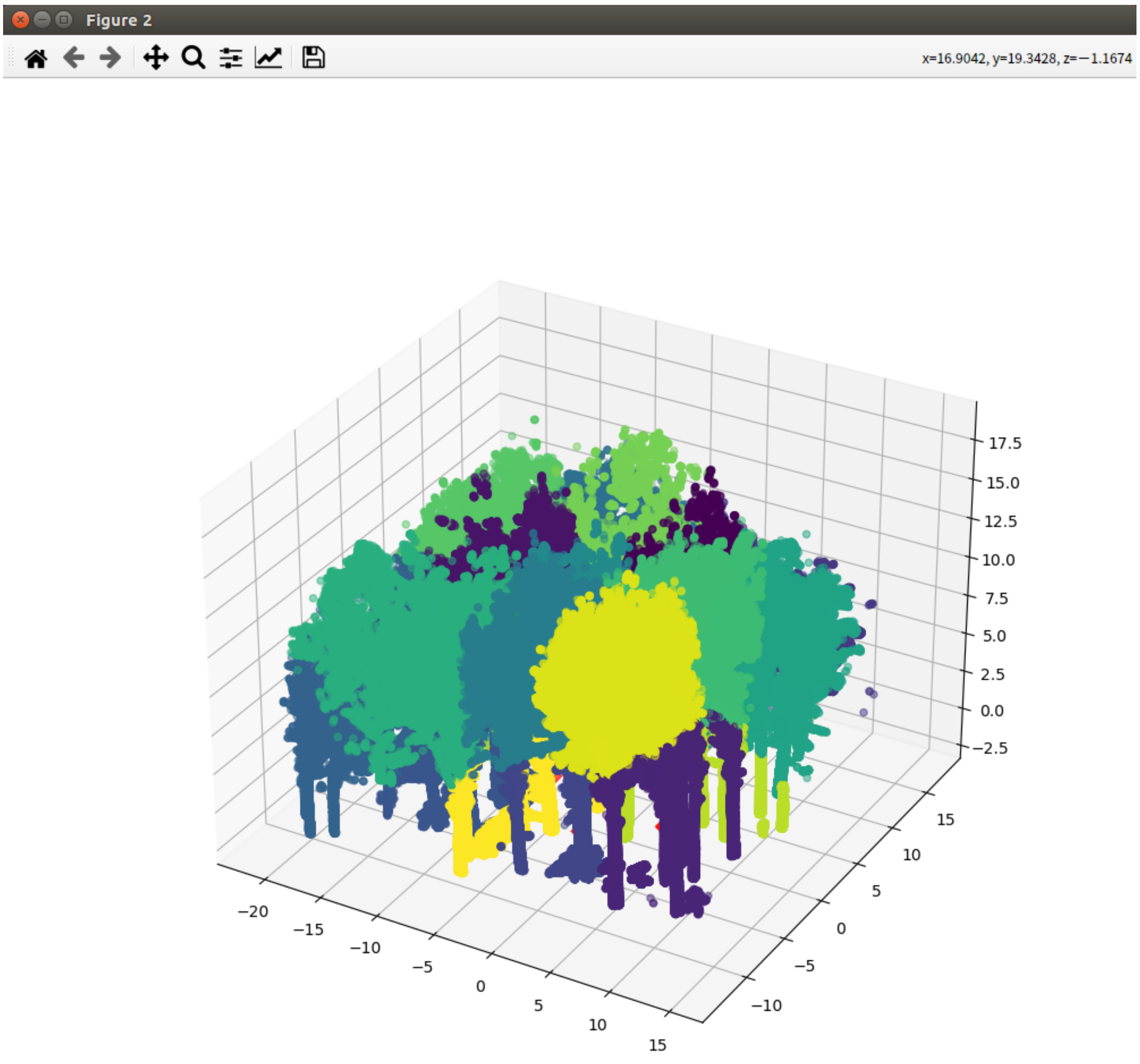


Figure 2
x=16.9042, y=19.3428, z=−1.1674
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−20
−15
−10
−5
0
5
10
15
15
10
5
0
−5
−10

문제점

정확한 k 를 알아야 군집화가 제대로 진행이 되는점.

시스템의 어느 시점부터 라벨링을 적용 해야 할 지에 대한 문제.(소프트웨어 설계 문제)

테스트 및 시간 문제, 특히 Inertias 를 적용했을 때 느려짐 문제

정확도 문제

진행 중인 사항

DBSCAN - 구현 예정

계층적 클러스터링 - 구현 중

참고: https://bcho.tistory.com/1203 [조대협의 블로그]