sklearn Clustering - 군집분석

```
import warnings
warnings.filterwarnings('ignore')
```

실습용 데이터 설정

1) Import Packages

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
```

2) Load Dataset

- Load iris Dataset

```
from sklearn.datasets import load_iris

iris = load_iris()
```

- pandas DataFrame

```
DF = pd.DataFrame(data = iris.data,
                  columns = ['sepal_length',
                             'sepal_width',
                             'petal_length',
                             'petal_width'])

DF.head(3)
```

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2

I. K-means

1) Modeling

```
from sklearn.cluster import KMeans

kmeans_3 = KMeans(n_clusters = 3,
                  init ='k-means++',
                  max_iter = 15,
                  random_state = 2045)

kmeans_3.fit(iris.data)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=15,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=2045, tol=0.0001, verbose=0)
```

2) 군집결과

```
DF['kmeans'] = kmeans_3.labels_
DF['target'] = iris.target
```

```
DF.groupby('target')['kmeans'].value_counts()
```

```
target  kmeans
0       1         50
1       0         48
        2          2
2       2         36
        0         14
Name: kmeans, dtype: int64
```

II. Mean Shift(평균 이동)

- 데이터의 분포를 이용하여 군집의 중심을 탐색
  - 데이터 밀도가 가장 높은 곳으로 중심을 지속적으로 이동
- 군집의 중심점은 데이터포인트가 모여있는 곳이라는 가정
  - 확률 밀도 함수(Probability Desity Function)
  - KDE(Kernel Density Estimation)

1) Modeling

- Hyperparameter
  - bandwidth : 대역폭이 클수록 적은 수의 군집중심을 가짐

```
from sklearn.cluster import MeanShift

meanshift = MeanShift(bandwidth = 0.86)

meanshift.fit_predict(iris.data)
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 2, 0, 2, 2, 2, 2, 0, 2, 2,
       2, 2, 2, 0, 0, 2, 2, 2, 2, 0, 2, 0, 2, 0, 2, 2, 0, 0, 2, 2, 2,
       2, 0, 2, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 2, 0, 2, 2, 0])
```

```
from sklearn.cluster import estimate_bandwidth

estimate_bandwidth(iris.data)
```

```
1.2020768127998687
```

2) 군집결과

```
DF['meanshift'] = meanshift.fit_predict(iris.data)
```

```
DF.groupby('target')['meanshift'].value_counts()
```

```
target  meanshift
0       1            50
1       0            48
        1             1
        2             1
2       2            36
        0            14
Name: meanshift, dtype: int64
```

III. GMM(Gaussian Mixture Model)

- 데이터가 여러 개의 가우시간 분포(Gaussian Distribution)를 가진 데이터들의 집합이라고 가정
  - 개별 정규분포의 모수(평균, 분산) 추정
  - 각 데이터포인트가 어떤 정규분포에 해당하는지 확률 추정
    - EM(Expectation and Maximization)

1) Modeling

```
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components = 3,
                      random_state = 2045)

gmm.fit(iris.data)
```

```
GaussianMixture(covariance_type='full', init_params='kmeans', max_iter=100,
                means_init=None, n_components=3, n_init=1, precisions_init=None,
                random_state=2045, reg_covar=1e-06, tol=0.001, verbose=0,
                verbose_interval=10, warm_start=False, weights_init=None)
```

```
gmm.predict(iris.data)
```

```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 2, 0, 2, 0, 2, 0, 0, 0, 0, 2, 0, 0, 0, 0, 0, 2, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

2) 군집결과

```
DF['gmm'] = gmm.predict(iris.data)
```

```
DF.groupby('target')['gmm'].value_counts()
```

```
target  gmm
0       1      50
1       0      45
        2       5
2       2      50
Name: gmm, dtype: int64
```

IV. DBSCAN(Density Based Spatial Clustering of Applications with Noise)

- 밀도(Density) 기반 군집
  - 기하학적으로 복잡한 데이터에도 효과적으로 군집 가능
  - 핵심 포인트(Core Point)들을 서로 연결하면서 군집화

1) Modeling

- Hyperparameter

- epsilon(esp, 입실론 주변 영역)
  - 개별 데이터포인트를 중심으로 '입실론 반경'을 가지는 주변 영역
  - 'Core Point' 기준
- min point(min_samples, 최소 데이터 개수)
  - 개별 데이터포인트의 '입실론 주변 영역'에 포함되는 다른 데이터포인트의 개수
  - 조건 만족 시 'Core Point'로 지정

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps = 0.8,
                min_samples = 8,
                metric = 'euclidean')

dbscan.fit_predict(iris.data)
```

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

2) 군집결과

```
DF['dbscan'] = dbscan.fit_predict(iris.data)
```

```
DF.groupby('target')['dbscan'].value_counts()
```

```
target  dbscan
0        0       50
1        1       50
2        1       47
        -1        3
Name: dbscan, dtype: int64
```

#

#

#

The End

#

#

#