프로젝트 기반 데이터 과학자 양성과정(Data Science)
Machine Learning 및 분석실습

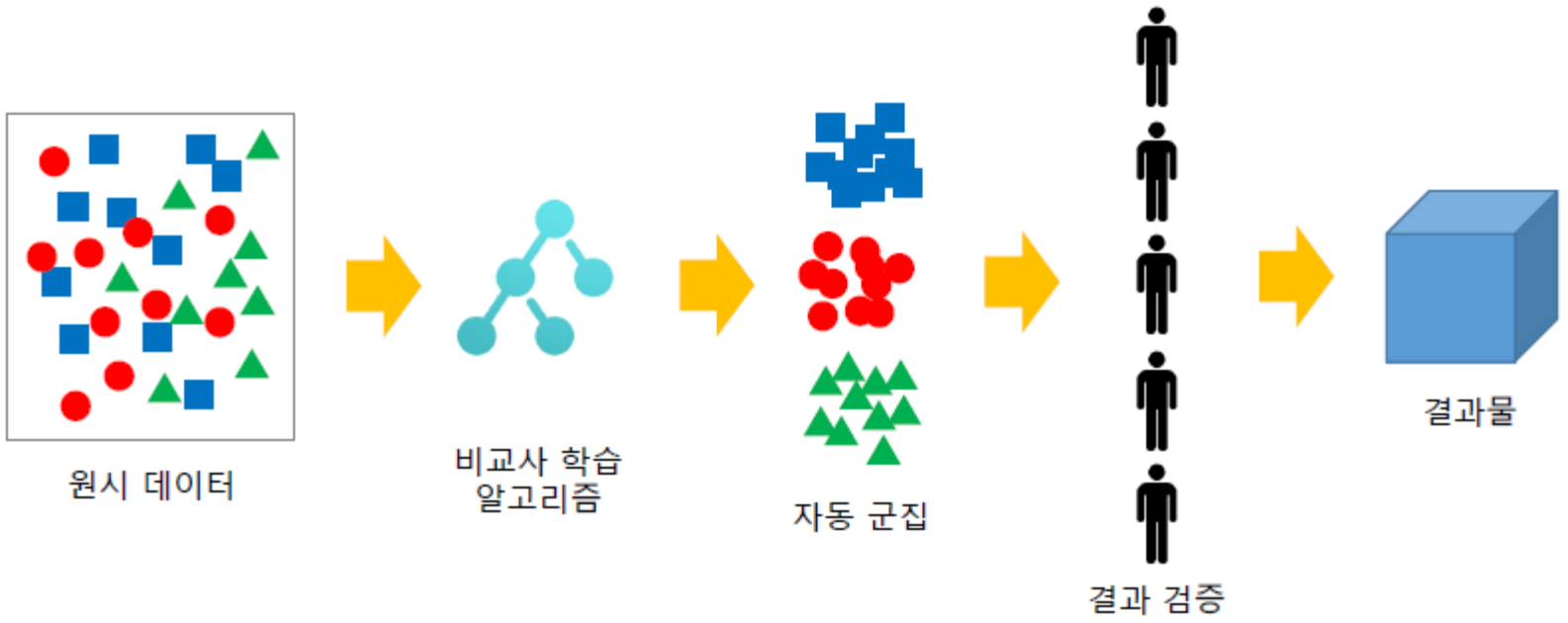
6주차
군집화 개요
K-means
hierarchical clustering
DBSCAN

강사 : 최영진

군집화 개요

❖ 비지도학습(Unsupervised Learning)

- 정답을 따로 알려주지 않고(label이 없음)
- 목표값을 정해주지 않아도 되고 사전 학습이 필요 없으므로 속도가 빠름
- 실무에서는 지도학습에서의 적절한 feature를 찾아내기 위한 전처리 방법으로 비지도 학습 사용



군집화 개요

❖ **비지도학습(Unsupervised Learning)**

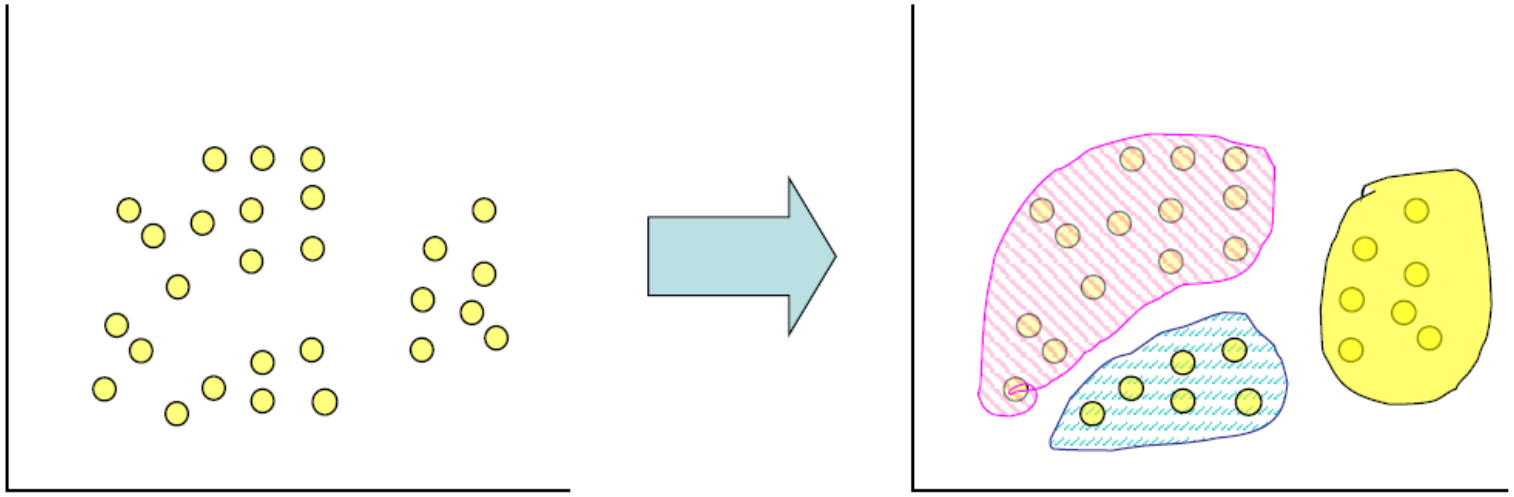
구분	알고리즘	설명
데이터 관계 측면	K-Means	임의의 중심점 기준 최소 거리 기반 군집화
	DBSCAN	반경 내 데이터 벡터 밀도 기반 군집화
특징 추출 측면	주성분 분석	사물의 주요 특징 분석 및 추출 차원 축소, 축 상의 투영으로 표시

군집화 개요

❖ 비지도학습(Unsupervised learning)

▪ 군집화

- 비슷한 특성을 가진 데이터들끼리 그룹화 같은 그룹내 요소들은 아주 유사하고 다른 그룹과는 확연히 다름

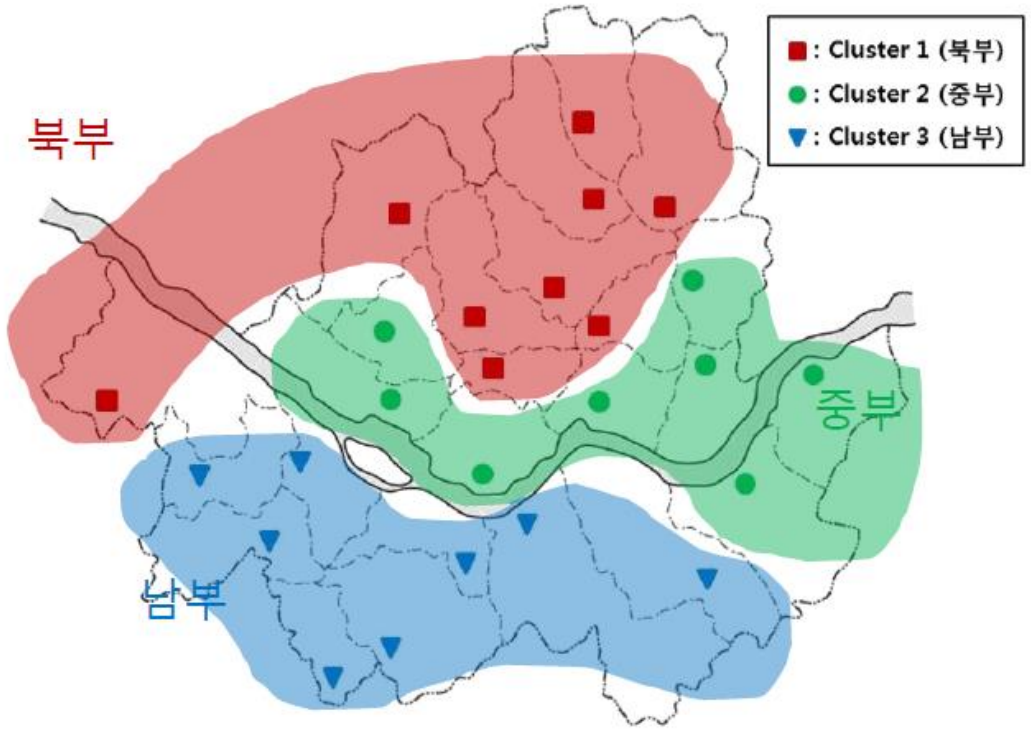


군집화 개요

❖ 비지도학습(Unsupervised learning)

▪ 군집화

- 비슷한 특성을 가진 데이터들끼리 그룹화 같은 그룹내 요소들은 아주 유사하고 다른 그룹과는 확연히 다름



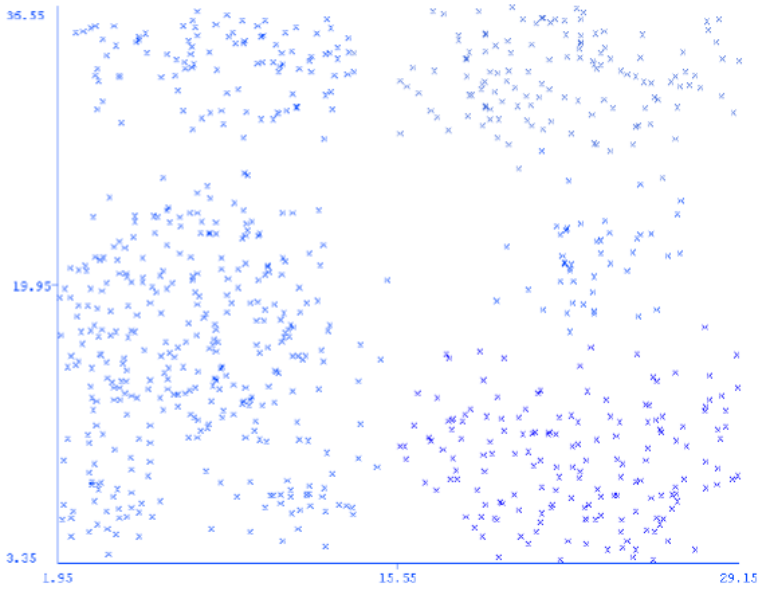
군집화 개요

❖ 비지도학습(Unsupervised learning)

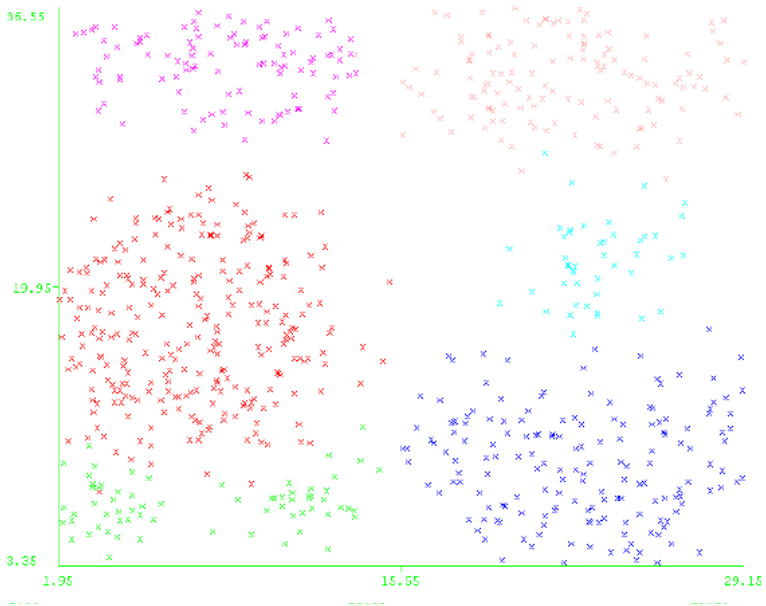
▪ 군집화 - k-means 군집

- 각 관측치가 가장 가까운 평균을 갖는 클러스터에 속하고 클러스터의 프로토타입 역할을 하는 k개의 클러스터로 n개의 관측치를 분할함

군집되지 않은 원본 데이터



군집된 데이터

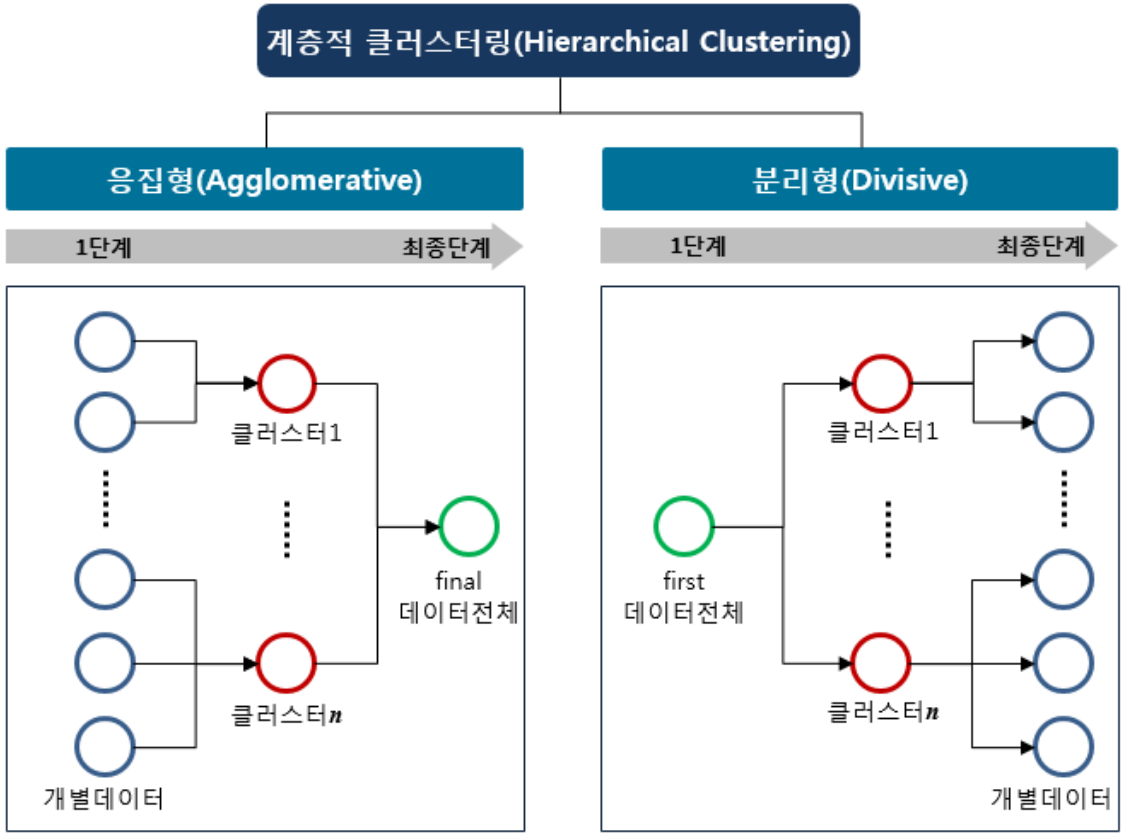


군집화 개요

❖ 비지도학습(Unsupervised learning)

▪ 군집화 – 계층적 군집화

- 특정 알고리즘에 의해 데이터를 연결하여 계층적 클러스터를 구성해 나가는 방법

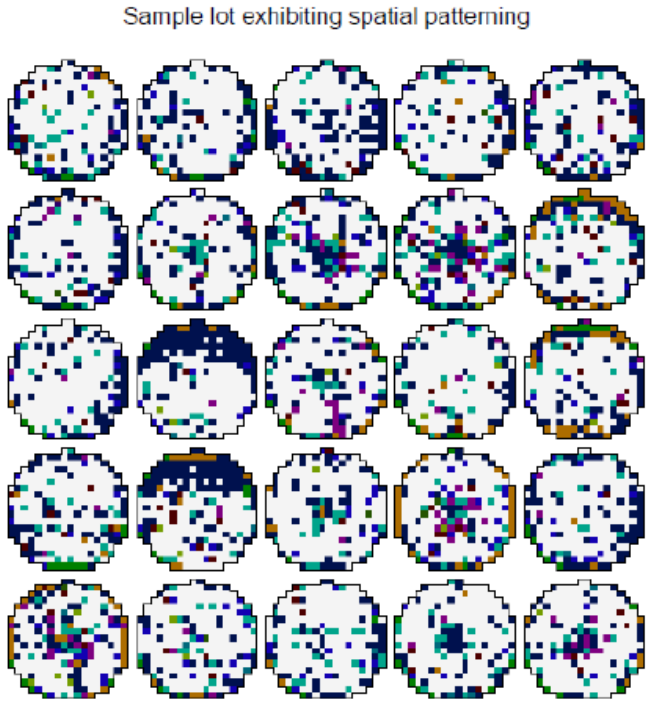


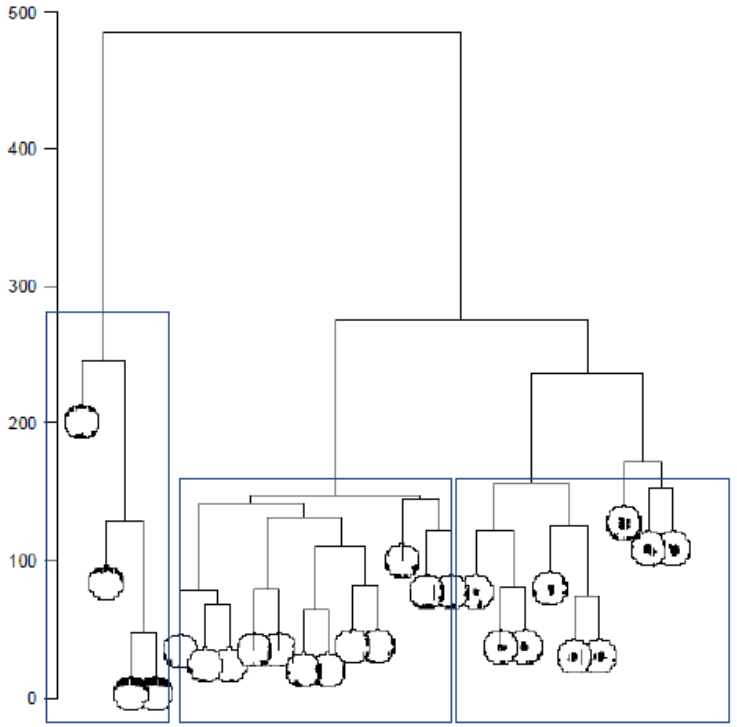
군집화 개요

❖ 비지도학습(Unsupervised learning)

▪ 군집화 – 계층적 군집화

- 웨이퍼 map 군집화를 통한 웨이퍼 불량 패턴 파악



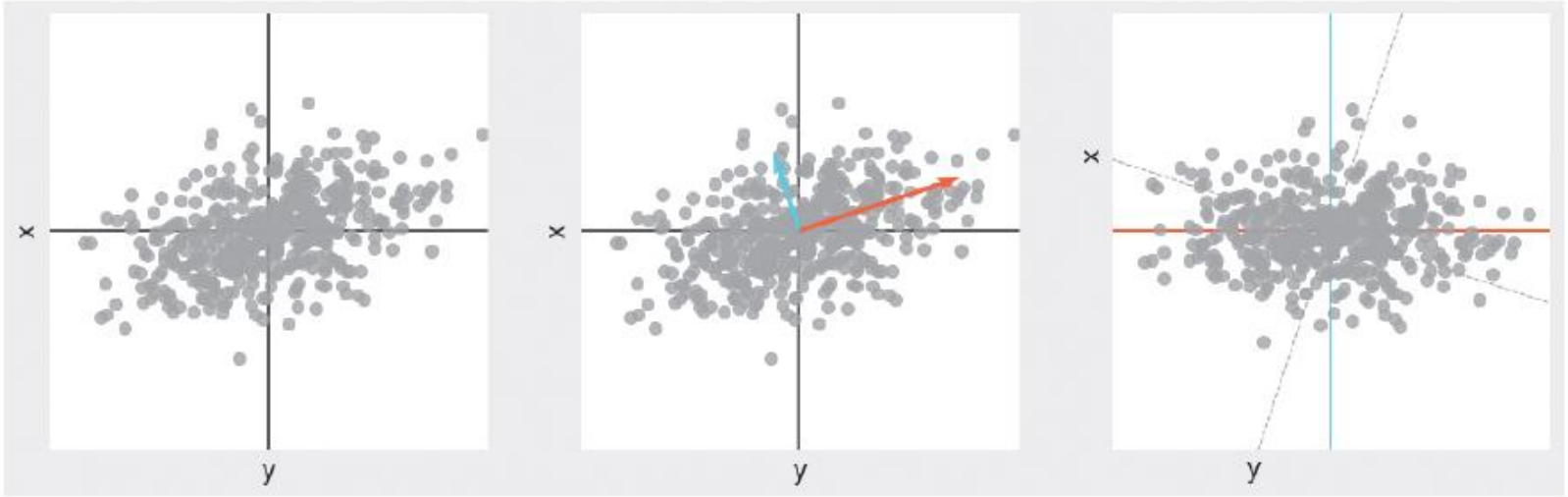


군집화 개요

❖ 비지도학습(Unsupervised learning)

▪ 주성분분석

- 수천개 또는 수백만개의 특징을 가지는 경우 차원 축소를 통해 반응변수를 가장 크게 변화시키는 특징을 추려내는 과정
- 데이터의 분산(variance)을 최대한 보존하면서 서로 직교하는 새 기저(축)를 찾아, 고차원 공간의 표본들을 선형 연관성이 없는 저차원 공간으로 변환하는 기법



군집화 개요

❖ 비지도학습(Unsupervised learning)

▪ 비지도 학습 알고리즘과 차원 축소 및 클러스터링 여부

번호	알고리즘 이름	차원 축소	클러스터링
10	주성분 분석(principal component analysis, PCA)	○	×
11	잠재 의미 분석(latent semantic analysis, LSA)	○	×
12	음수 미포함 행렬 분해(non-negative matrix factorization, NMF)	○	×
13	잠재 디리클레 할당(latent Dirichlet allocation, LDA)	○	×
14	k-평균 알고리즘(k-means algorithm)	×	○
15	가우시안 혼합 모델(Gaussian mixture model)	×	○
16	국소 선형 임베딩(local linear embedding, LLE)	○	×
17	t-분포 확률적 임베딩(t-distributed stochastic neighbor embedding, t-SNE)	○	×

군집화 개요

❖ **비지도학습(Unsupervised learning)**

구분	지도학습	비지도학습
사용이유	예측 모델 생성	고차원 데이터 분류
성능평가	교차 검증 수행	검증 방법 없음
입력정보	Labeled Data	Raw Data
유형	회귀: (x, y)로 f(x)=y파악 분류: 그룹별 특징 파악	군집: 데이터끼리 묶음 패턴인식: 여러 그룹 인식
알고리즘	머신러닝	K-Means, DBSCAN, 군집(Clustering) 등
사례	패턴인식, 질병진단 주가 예측, 회귀 분석	스팸필터, 차원 축소 데이터마이닝, 지식발굴

군집화 개요

❖ 비지도학습(Unsupervised learning)

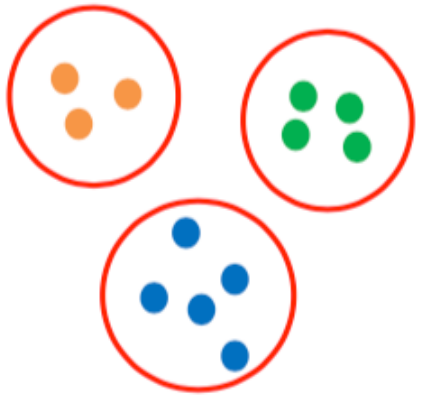
▪ 분류와 군집화의 차이

구분	Classification (분류)	Clustering (군집화)
정보	O	X
Label 유무	O	X
종류	Supervised Learning (지도 학습)	Unsupervised Learning (비지도 학습)
공통점	데이터를 비슷한 집단으로 묶는 방법	

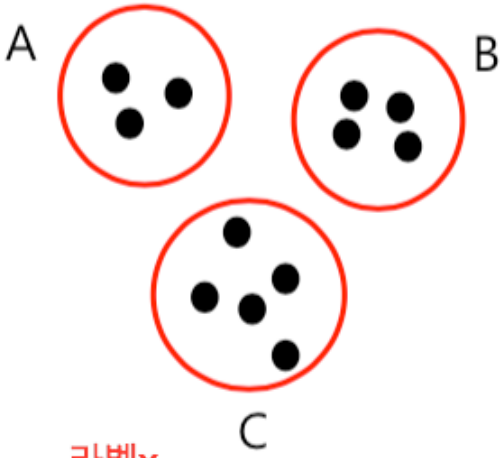
군집화 개요

❖ 비지도학습(Unsupervised learning)

- **분류와 군집화의 차이**
- **'분류'란, 소속집단의 정보를 이미 알고 있는 상태에서, 비슷한 집단으로 묶는 방법**
  - **Label 이 있는 data를 나누는 방법으로, Supervised Learning (지도학습)**
- **'군집화'란, 소속집단의 정보가 없고, 모르는 상태에서, 비슷한 집단으로 묶는 방법**
  - **Label이 없는 data를 군집단위로 나누는 것으로, Unsupervised Learning (비지도학습)**



라벨0

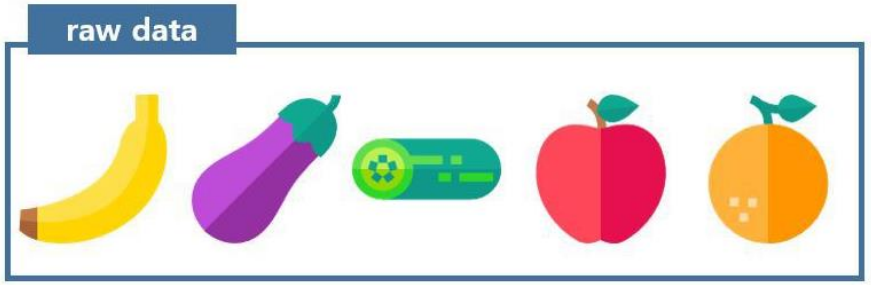


라벨x

군집화 개요

❖ 비지도학습(Unsupervised learning)

▪ 분류와 군집화의 차이



군집화 개요

❖ 군집화 알고리즘

- **K-Means Clustering**
- **Hierarchical Clustering**
- **DBSCAN (Density-Based Spatial Clustering of Applications with Noise)**

군집화 개요

❖ 군집화 알고리즘

- **K-Means Clustering**
  - 비계층적 군집화 - 분리형 군집화
  - 전체 데이터의 영역을 특정 기준에 의해 동시에 구분
  - 각 개체들은 사전에 정의된 개수의 군집 중 하나에 속하게 됨

- **Hierarchical Clustering**
  - 계층적 군집화
  - 개체들을 가까운 집단부터 차근차근 묶는 방식
  - 군집화 결과 뿐만 아니라 유사한 개체들이 결합되는 Dendrogram도 생성

- **DBSCAN**
  - 비계층적 군집화 - 분포 기반 군집화
  - 데이터의 분포를 기반으로 높은 밀도를 갖는 세부 영역들로 전체 영역 구분

군집화

❖ k-평균 알고리즘(K-means algorithm)

- **주어진 데이터를 k개의 클러스터로 묶는 알고리즘으로, 각 클러스터와 거리 차이의 분산을 최소화하는 방식**
- 레이블이 달려 있지 않은 입력 데이터에 레이블을 달아주는 역할을 수행
- 각 군집 중심의 위치 (2)각 개체가 어떤 군집에 속해야 하는지 EM 알고리즘을 이용
  - 데이터 분류 , 클러스터링 방법
  - 성향이 불분명한 시장 분석
  - 패턴인식, 음성인식 기본
  - 관련성 파악 시 사용
- "**K**"는 데이터 세트에서 찾을 것으로 예상되는 클러스터(그룹) 수
- "**Means**"는 각 데이터로부터 그 데이터가 속한 클러스터의 중심까지의 평균 거리

군집화

❖ k-평균 알고리즘(K-means algorithm)

- **장점**
  - 간단한 알고리즘으로 대규모에도 적용이 가능. (계산시간이 짧다)
  - 데이터에 대한 사전정보가 필요 없음/ 특정 변수에 대한 역할 정의 필요 없음

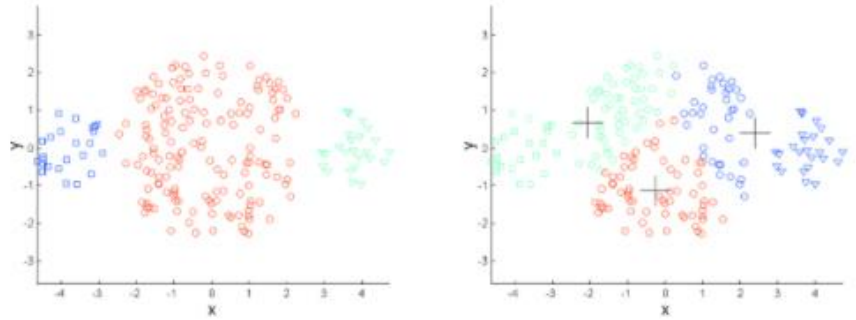
- **단점**
  - K개를 정하는데 어려움
  - 결과해석이 어려운 단점(분석 결과가 관찰치 사이의 거리 또는 유사성을 어떻게 정의하느냐에 따라 크게 좌우)
  - 초기 클러스터링 개수 선정

군집화

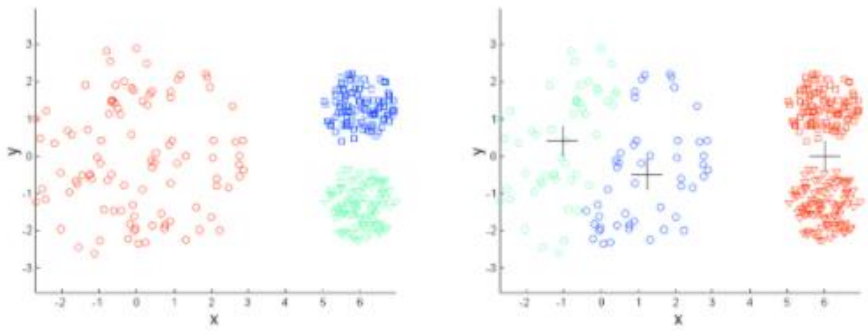
❖ k-평균 알고리즘(K-means algorithm)

▪ 단점

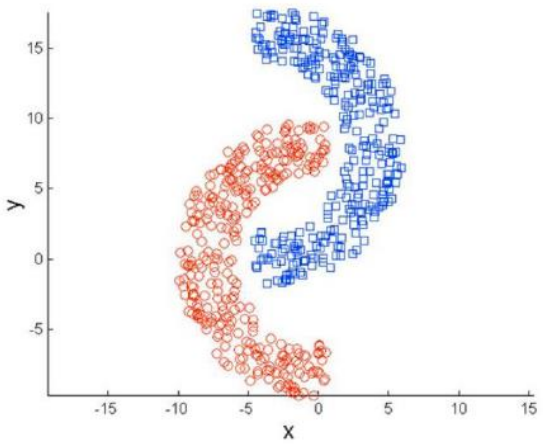
군집의 크기가 다를 경우 제대로 작동하지 않을 수 있습니다.



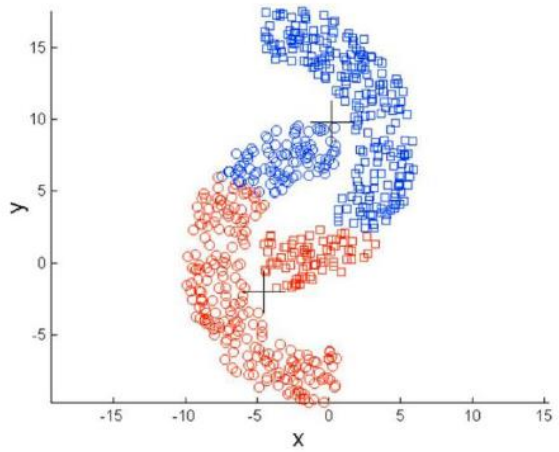
군집의 밀도가 다를 경우에도 마찬가지입니다.



정답



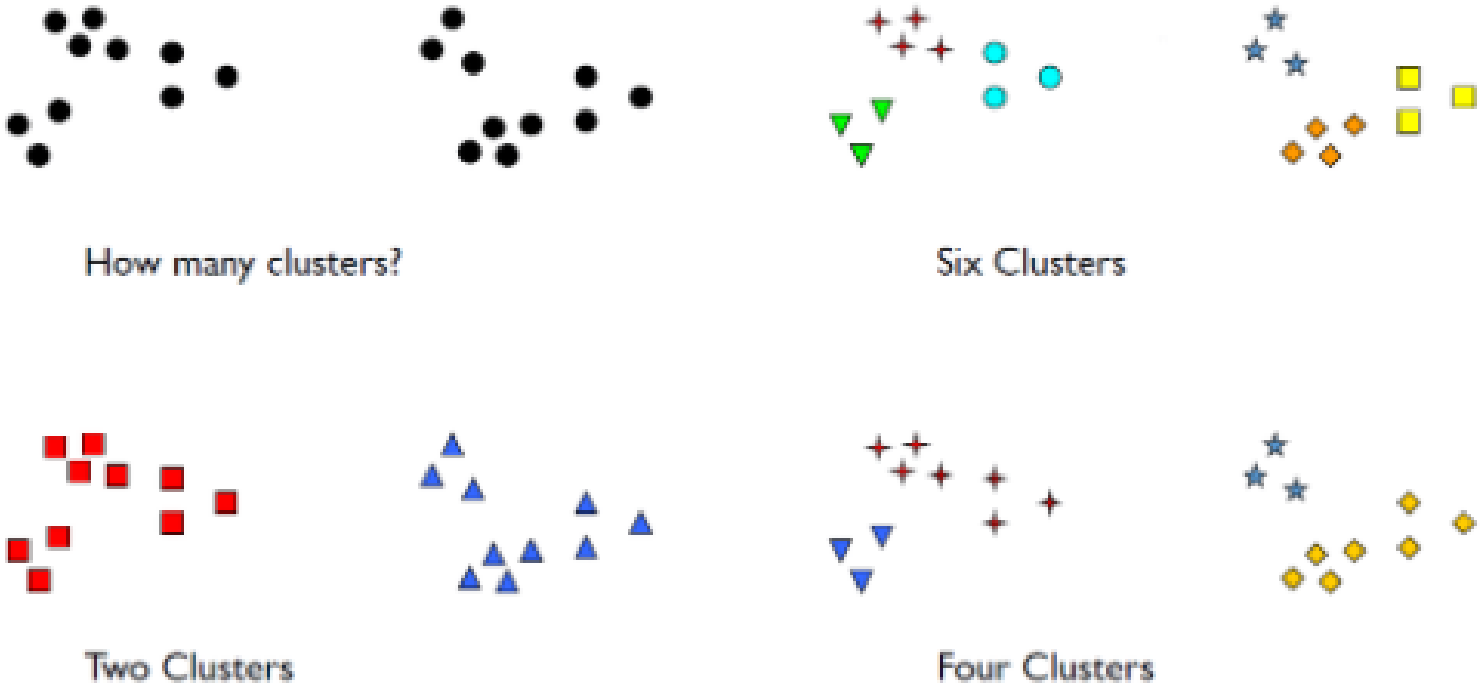
K-평균 군집화 결과



군집화

❖ **k-평균 알고리즘**(K-means algorithm)

- **정답이 없기 때문에 일반적인 머신러닝 알고리즘처럼 단순정확도(Accuracy) 등 지표로 평가할 수 없음**



군집화

❖ k-평균 알고리즘(K-means algorithm)

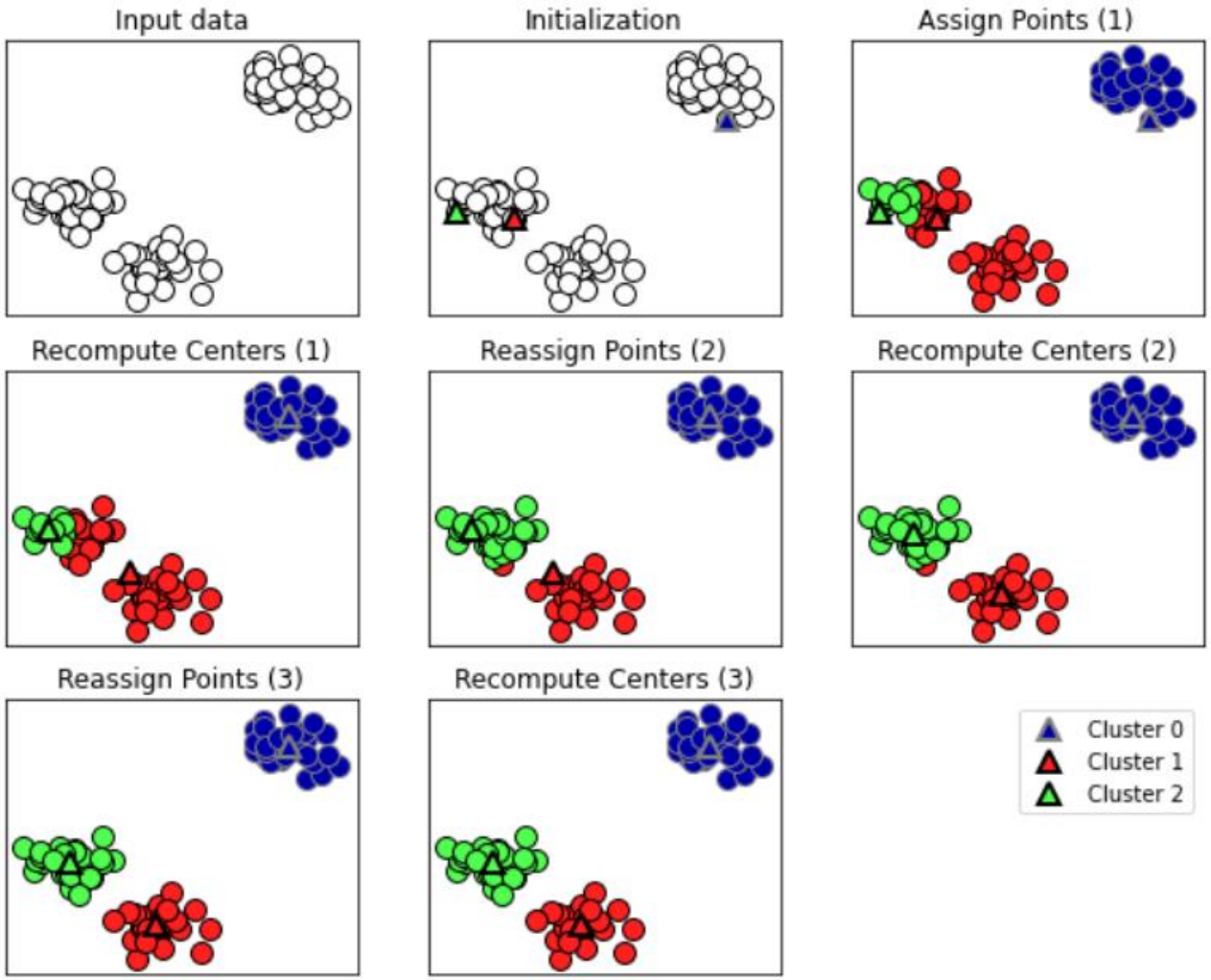
- **1) 초기 *k* "평균값" (위의 경우 *k*=3) 은 데이터 오브젝트 중에서 무작위로 뽑힘. (색칠된 동그라미로 표시됨).**
- **2) *k* 각 데이터 오브젝트들은 가장 가까이 있는 평균값을 기준으로 묶이고 평균값을 기준으로 분할된 영역은 다이어그램으로 표시**
- **3) *k*개의 클러스터의 중심점 기준으로 평균값이 재조정.**
- **4) 수렴할 때까지 2), 3) 과정을 반복**

표준 알고리즘의 실행 과정

군집화

❖ **k-평균 알고리즘**(K-means algorithm)

-

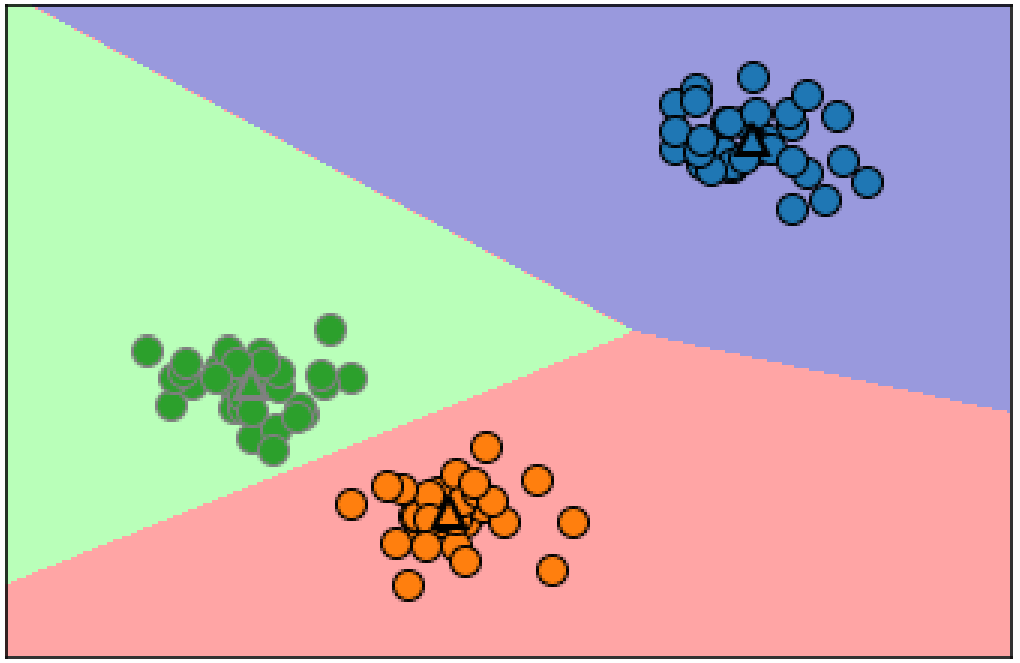


초기화
포인트 할당
중심재계산

군집화

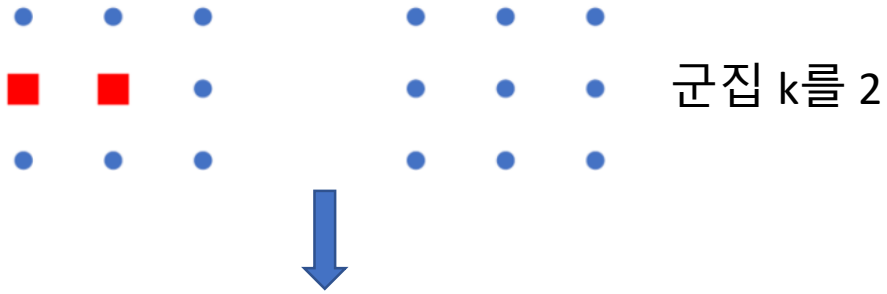
❖ **k-평균 알고리즘**(K-means algorithm)

- **mglearn.plots.plot_kmeans_boundaries()**
-



군집화

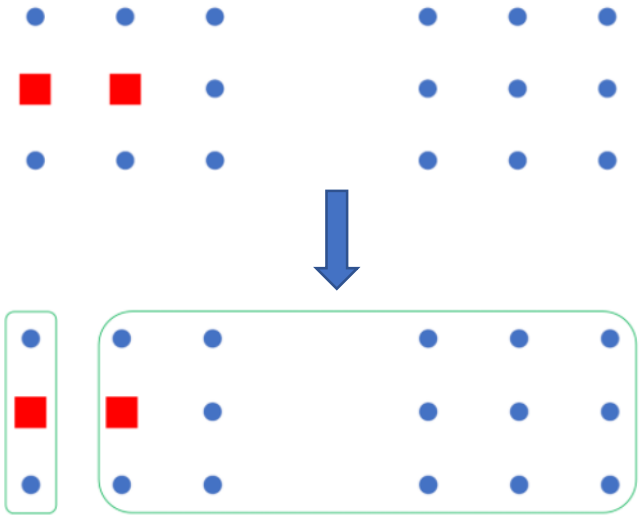
❖ **k-평균 알고리즘**(K-means algorithm)



군집 k를 2

군집화

❖ **k-평균 알고리즘**(K-means algorithm)



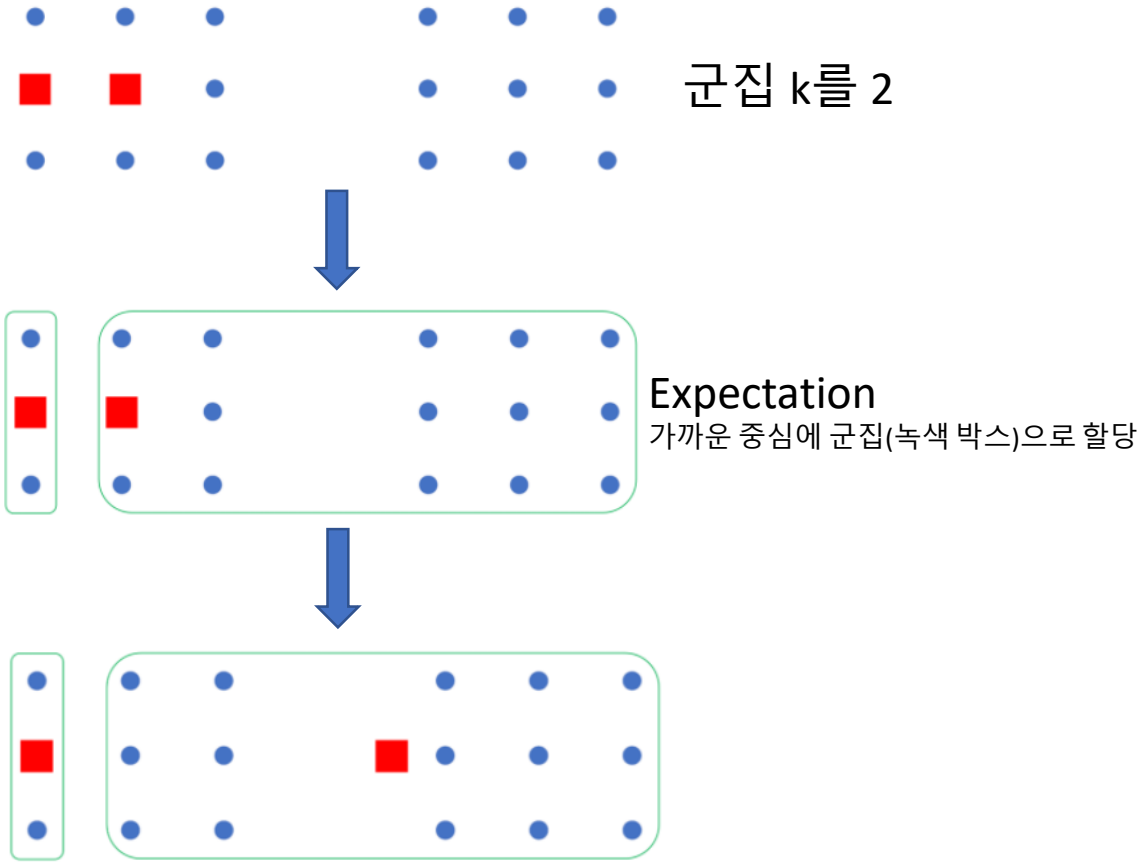
군집 k를 2

Expectation

가까운 중심에 군집(녹색 박스)으로 할당

군집화

❖ **k-평균 알고리즘**(K-means algorithm)



군집 k를 2

Expectation

가까운 중심에 군집(녹색 박스)으로 할당

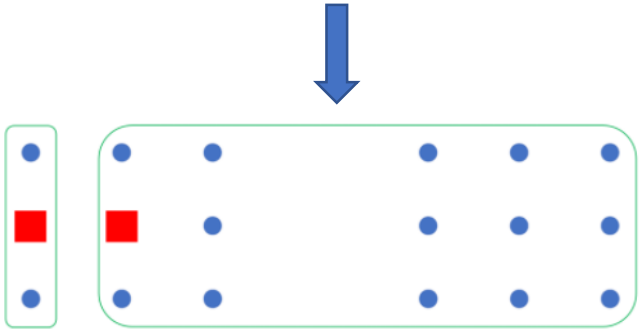
Maximization

군집 경계에 맞게 업데이트

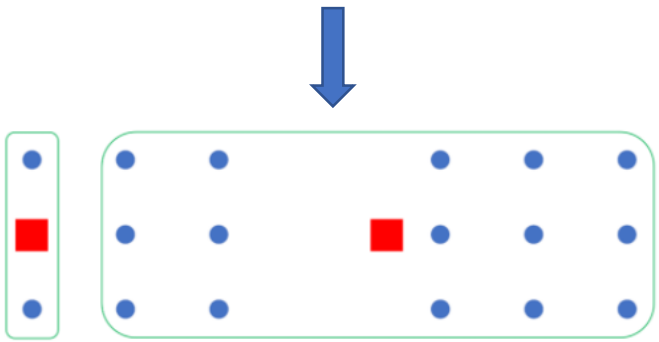
군집화

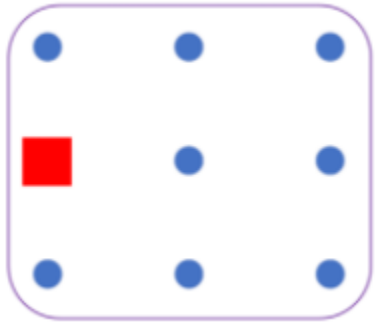
❖ **k-평균 알고리즘**(K-means algorithm)

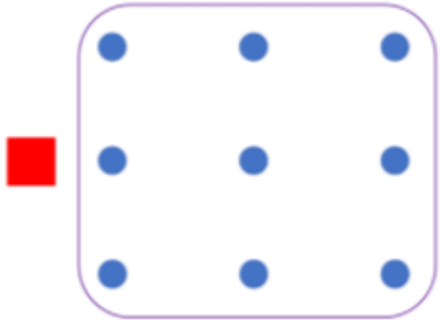
군집 k를 2



Expectation
가까운 중심에 군집(녹색 박스)으로 할당





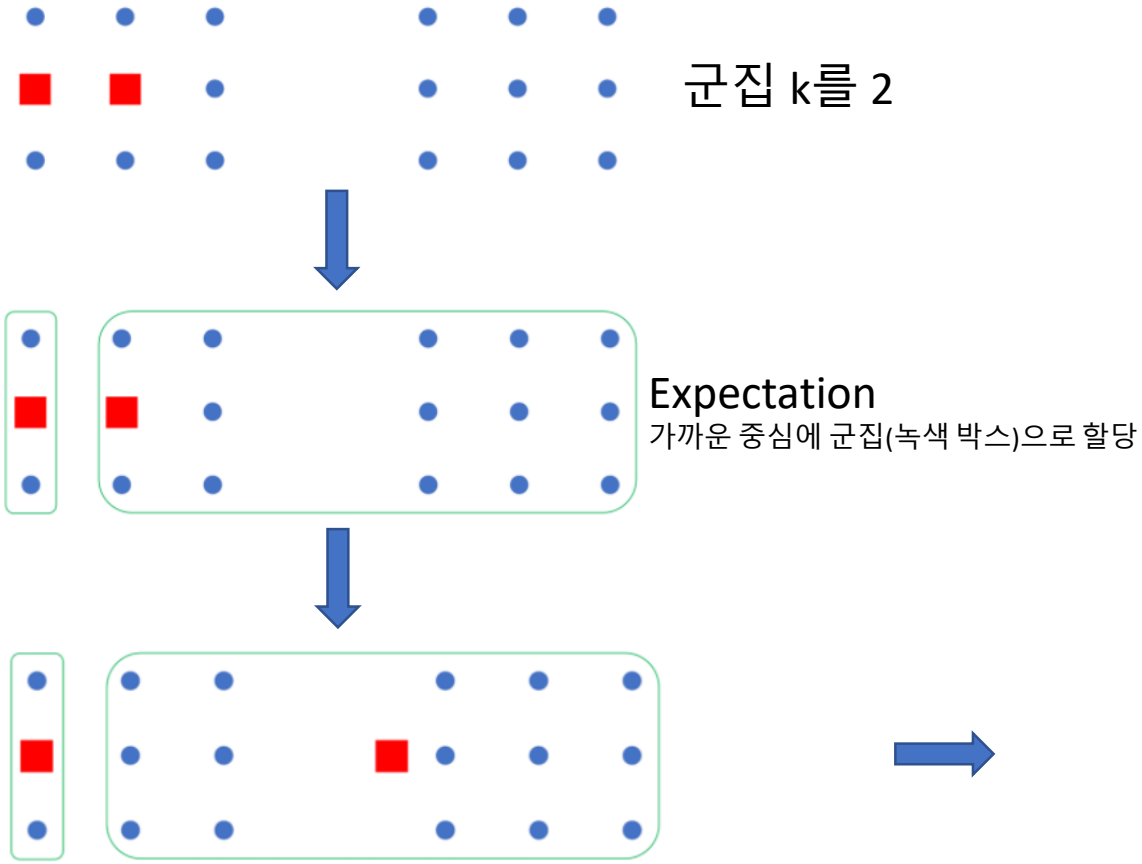


Maximization
군집 경계에 맞게 업데이트

Expectation
가장 가까운 중심에 군집(보라색 박스)으로 할당

군집화

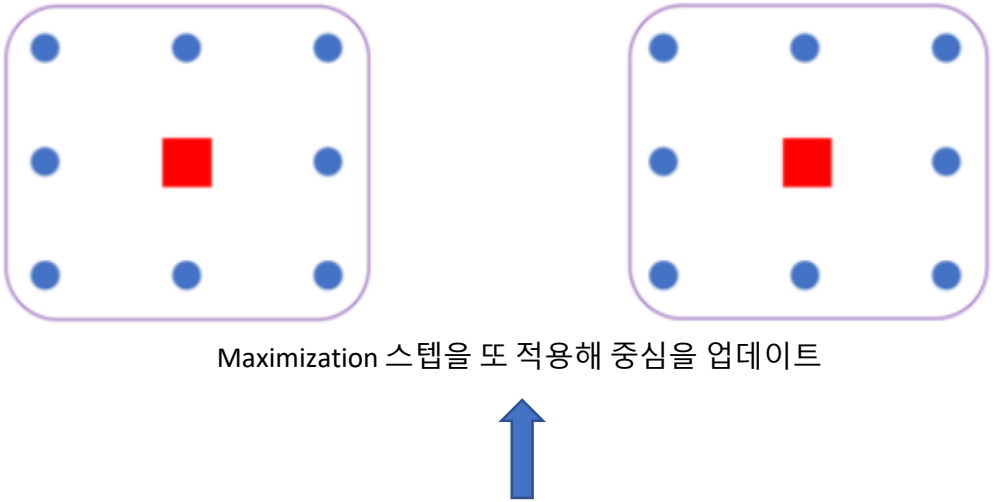
❖ **k-평균 알고리즘**(K-means algorithm)



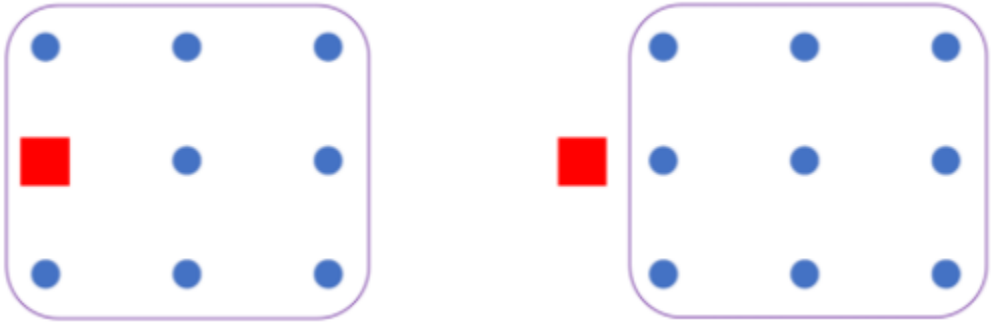
군집 k를 2

Expectation
가까운 중심에 군집(녹색 박스)으로 할당

Maximization
군집 경계에 맞게 업데이트



Maximization 스텝을 또 적용해 중심을 업데이트



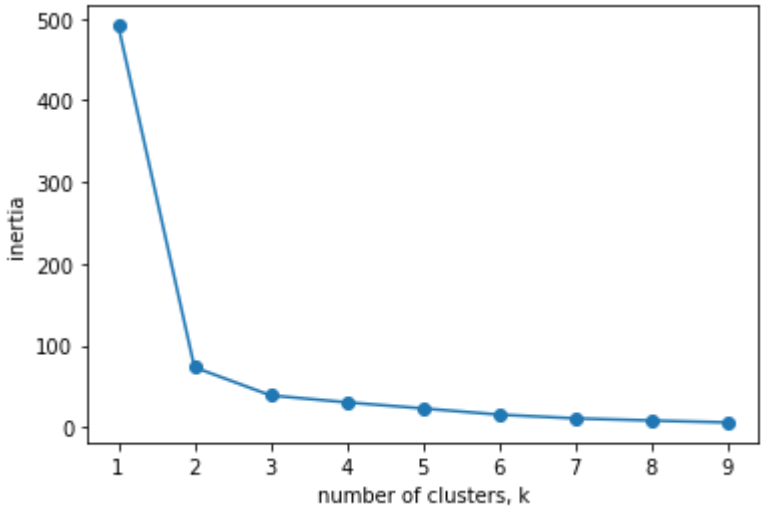
Expectation
가장 가까운 중심에 군집(보라색 박스)으로 할당

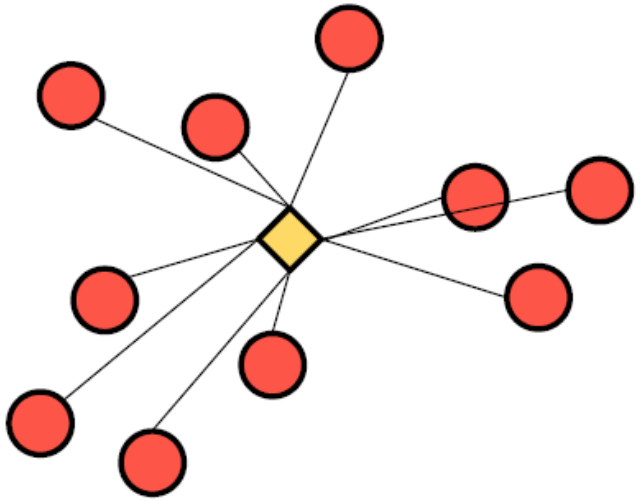
군집화

❖ **k-평균 알고리즘**(K-means algorithm) 결과 측정 및 평가

- **Sum of Squared Error (SSE)**

$$SSE = \sum_{i=1}^{K} \sum_{x \in C_i} dist(x, c_i)^2$$





: 관측치 (x)

: 중심 (C_i)

군집화

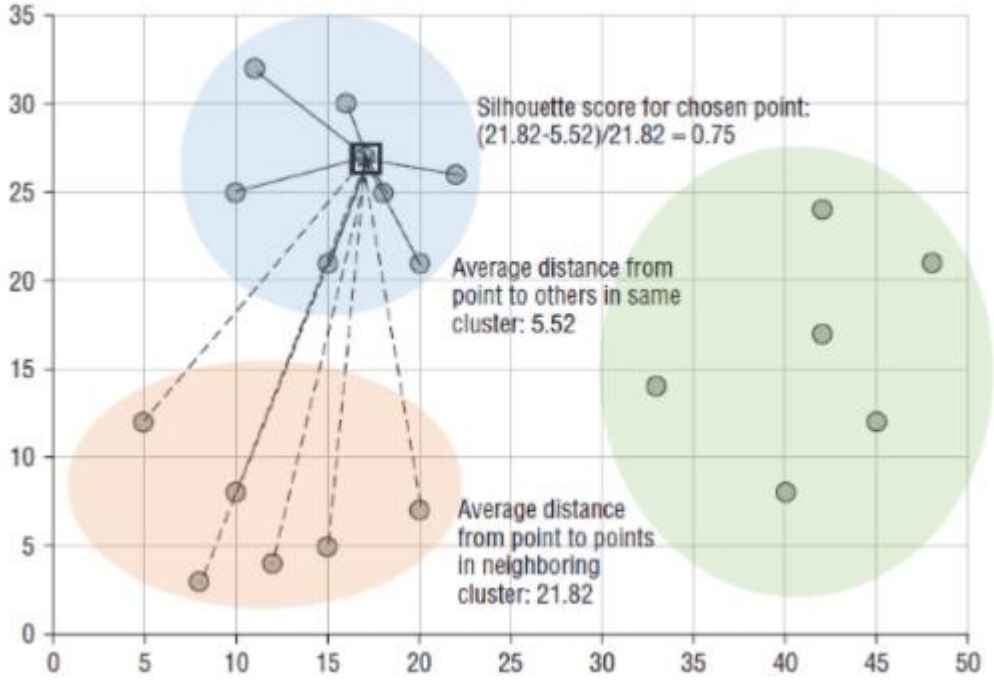
❖ **k-평균 알고리즘**(K-means algorithm) 결과 측정 및 평가

- **Silhouette 통계량**

$$s(i) = \frac{b(i) - a(i)}{\max\{a(i), b(i)\}}$$

- a(i)는 i번째 개체와 같은 군집에 속한 요소들 간 거리들의 평균
- b(i)는 i번째 개체와 다른 군집에 속한 요소들 간 거리들의 평균

- -1 ~ 1 사이의 값 : 0.5 이상일 경우 cluster good!



군집화

❖ k-평균 알고리즘(K-means algorithm)

- **from sklearn.cluster import KMeans**
- **model = KMeans(n_clusters=k)**
- **model.fit(data)**
- **model.predict(samples)**
- **model.inertia_**
- 얼마나 퍼져 있는지 (혹은 얼마나 뭉쳐있는지) 응집도는 **inertia** 값으로 확인
  inertia는 각 데이터로부터 자신이 속한 군집의 중심까지의 거리를 의미하기 때문에
  inertia 값이 낮을수록 군집화가 더 잘됨

군집화

❖ **k-평균 알고리즘**(K-means algorithm)

- **n_clusters: 군집의 갯수**
- **init: 초기화 방법. "random"이면 무작위, 각 데이터의 군집 라벨.**
- **n_init: 초기 중심위치 시도 횟수. 디폴트는 10이고 10개의 무작위 중심위치 목록 중 가장 좋은 값을 선택**
- **max_iter: 최대 반복 횟수.**
- **random_state: 시드값.**

군집화

❖ k-평균 알고리즘(K-means algorithm)

```
import numpy as np
from sklearn.cluster import KMeans

##########데이터 로드

x_data = np.array([
    [2, 1],
    [3, 2],
    [3, 4],
    [6, 5],
    [7, 5],
    [2, 5],
    [9, 2],
    [6, 3],
    [2, 5]
])

```

```
from sklearn.metrics import accuracy_score

y_data = np.array([1, 1, 2, 0, 0, 2, 1, 0, 2])

model = KMeans(n_clusters=3, random_state=4)
model.fit(x_data)

print(model.labels_)
print("정확도: {:.2f}".format(accuracy_score(y_data, model.labels_)))
```

```
[1 1 2 0 0 2 0 0 2]
정확도: 0.89
```

군집화

❖ k-평균 알고리즘(K-means algorithm)

```
1 import pandas as pd
2
3 x_data=pd.DataFrame(x_data, columns=['x','y'])
4 print(x_data)
5 x_data['cluster']=model.labels_
6 print(x_data)
```

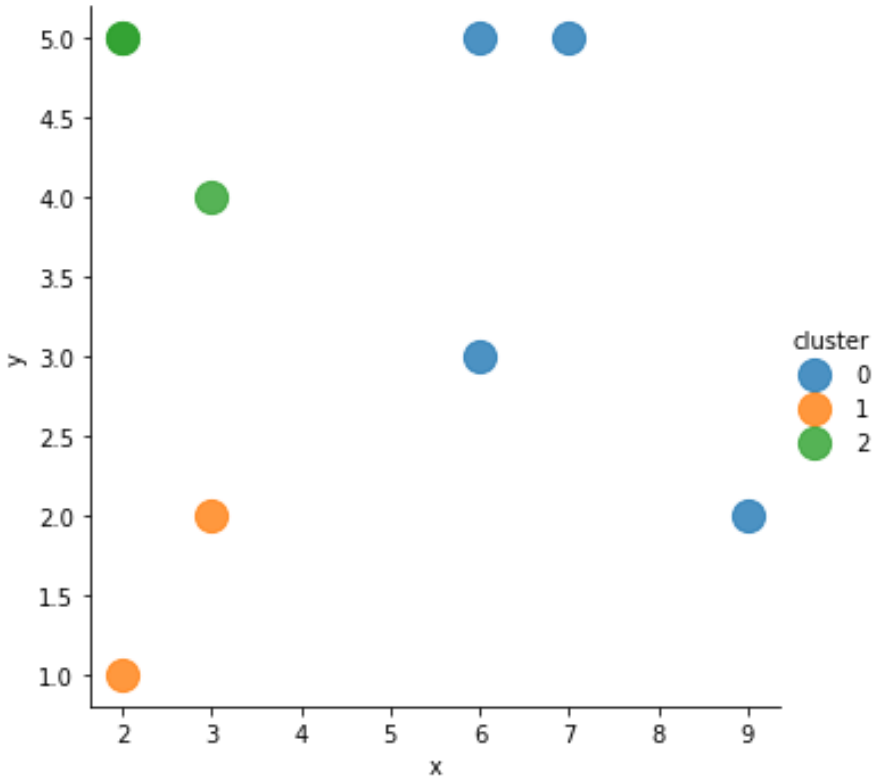
```
   x  y
0  2  1
1  3  2
2  3  4
3  6  5
4  7  5
5  2  5
6  9  2
7  6  3
8  2  5
   x  y  cluster
0  2  1        0
1  3  2        0
2  3  4        2
3  6  5        1
4  7  5        1
5  2  5        2
6  9  2        1
7  6  3        1
8  2  5        2
```

군집화

❖ **k-평균 알고리즘**(K-means algorithm)

```
1 import seaborn as sns
2 sns.lmplot('x','y',data=x_data, fit_reg=False, scatter_kws={"s": 200}, hue="cluster")
```

```
<seaborn.axisgrid.FacetGrid at 0x2d559e01470>
```

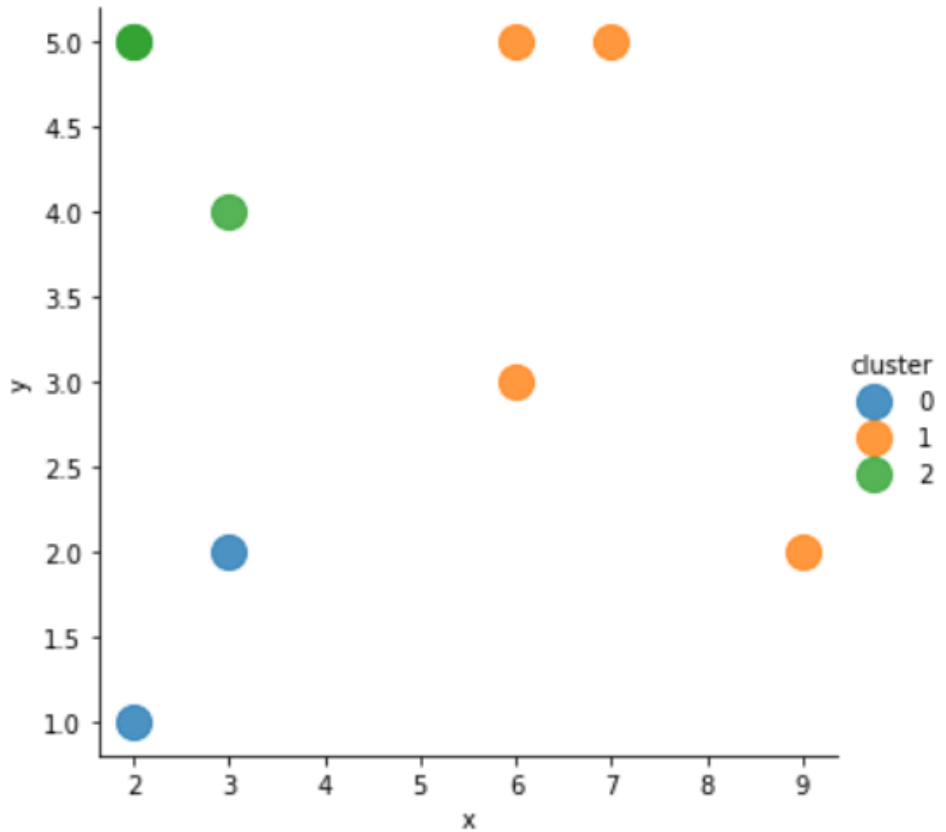


군집화

❖ **k-평균 알고리즘**(K-means algorithm)

```
1  import seaborn as sns
2  sns.lmplot('x','y',data=x_data, fit_reg=False, scatter_kws={"s": 200}, hue="cluster")
```

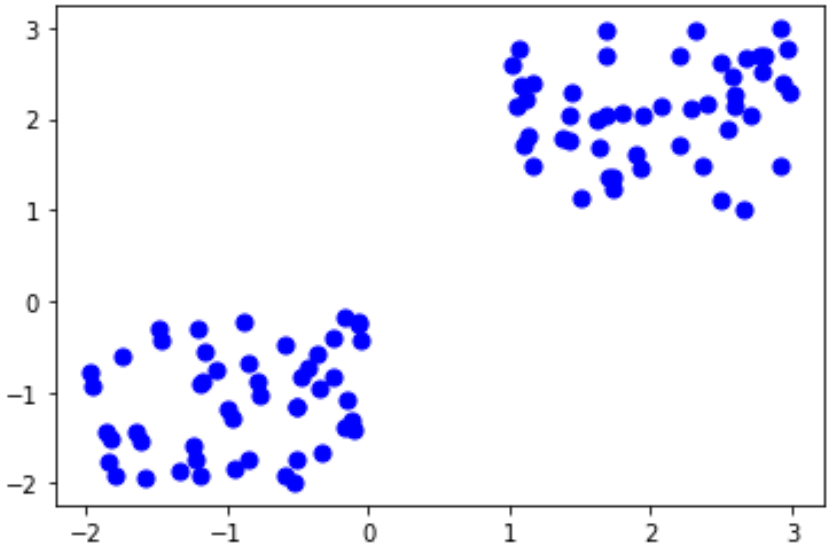
```
<seaborn.axisgrid.FacetGrid at 0x2d556fa7470>
```



군집화

❖ **k-평균 알고리즘**(K-means algorithm)

```
import random
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans
%matplotlib inline
random.seed(42)
X= -2 * np.random.rand(100,2)
X1 = 1 + 2 * np.random.rand(50,2)
X[50:100, :] = X1
plt.scatter(X[ : , 0], X[ :, 1], s = 50, c = 'b')
plt.show()
```



군집화

❖ **k-평균 알고리즘**(K-means algorithm)

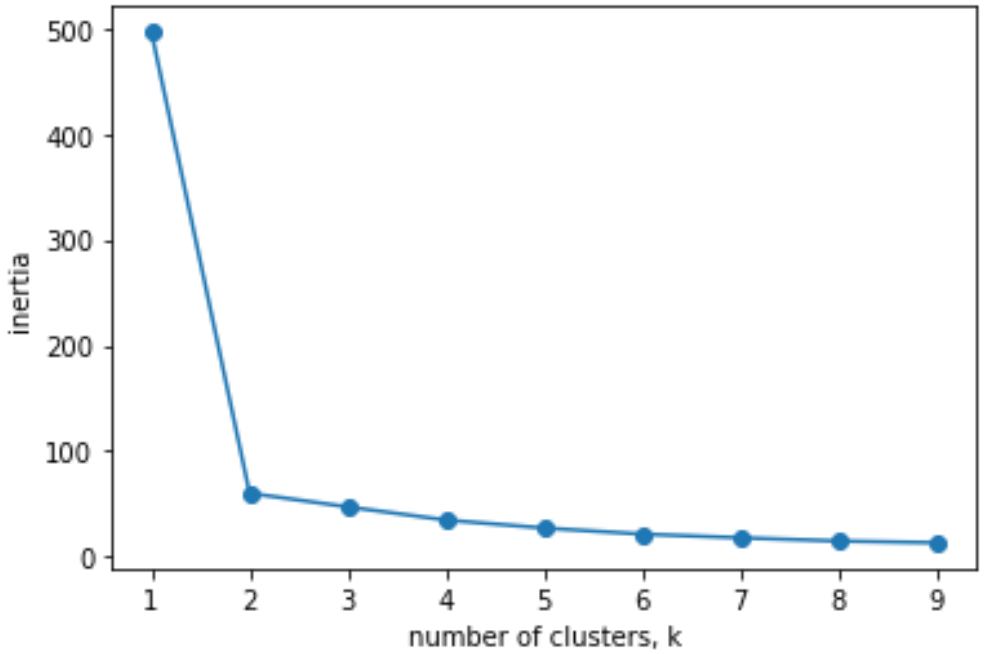
```
1 from sklearn.cluster import KMeans
2 Kmean = KMeans(n_clusters=2)
3 Kmean.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

군집화

❖ **k-평균 알고리즘**(K-means algorithm)

```
ks = range(1,10)
inertias = []

for k in ks:

    model = KMeans(n_clusters=k)
    model.fit(X)
    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()

```



군집화

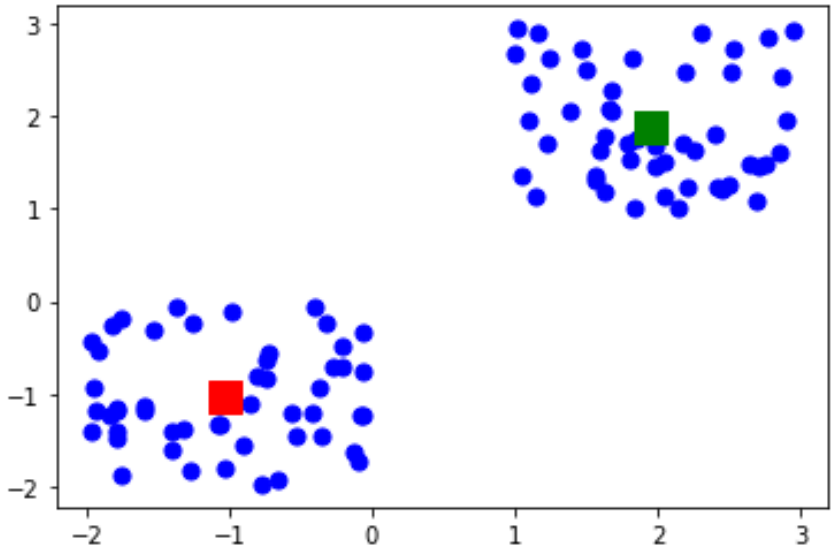
❖ **k-평균 알고리즘**(K-means algorithm)

```
1  Kmean.cluster_centers_
```

```
array([[ 1.95705186,  1.87714076],
       [-1.02445812, -1.03168951]])
```

```
1
```

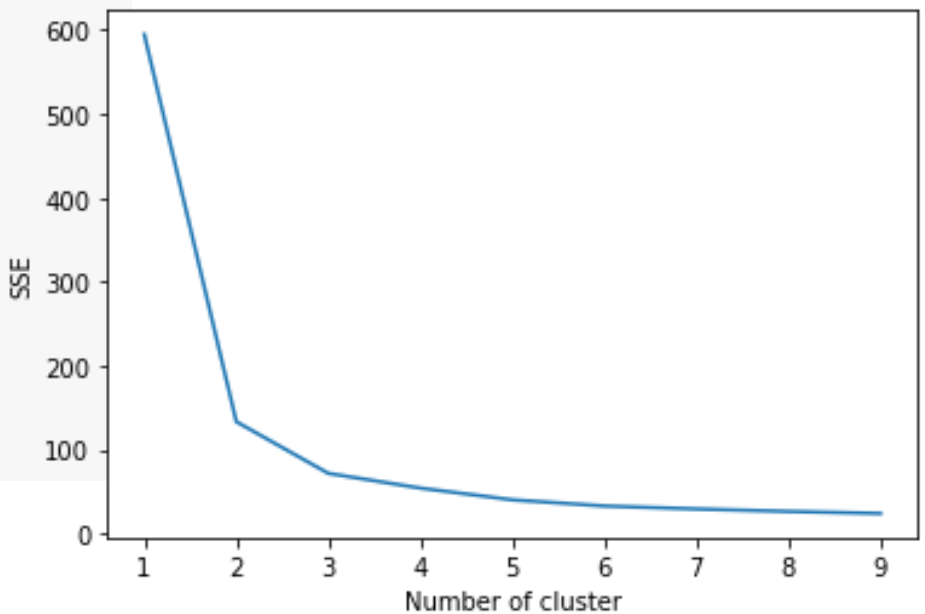
```
1  plt.scatter(X[ : , 0], X[ : , 1], s =50, c='b')
2  plt.scatter(1.95705186,  1.87714076, s=200, c='g', marker='s')
3  plt.scatter(-1.02445812, -1.03168951, s=200, c='r', marker='s')
4  plt.show()
```



군집화

❖ **k-평균 알고리즘**(K-means algorithm)

```
import pandas as pd
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

iris = load_iris()
X = pd.DataFrame(iris.data, columns=iris['feature_names'])
#print(X)
data = X[['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)']]

sse = {}
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, max_iter=1000).fit(data)
    data["clusters"] = kmeans.labels_
    sse[k] = kmeans.inertia_
plt.figure()
plt.plot(list(sse.keys()), list(sse.values()))
plt.xlabel("Number of cluster")
plt.ylabel("SSE")
plt.show()
```



군집화

❖ **k-평균 알고리즘**(K-means algorithm)

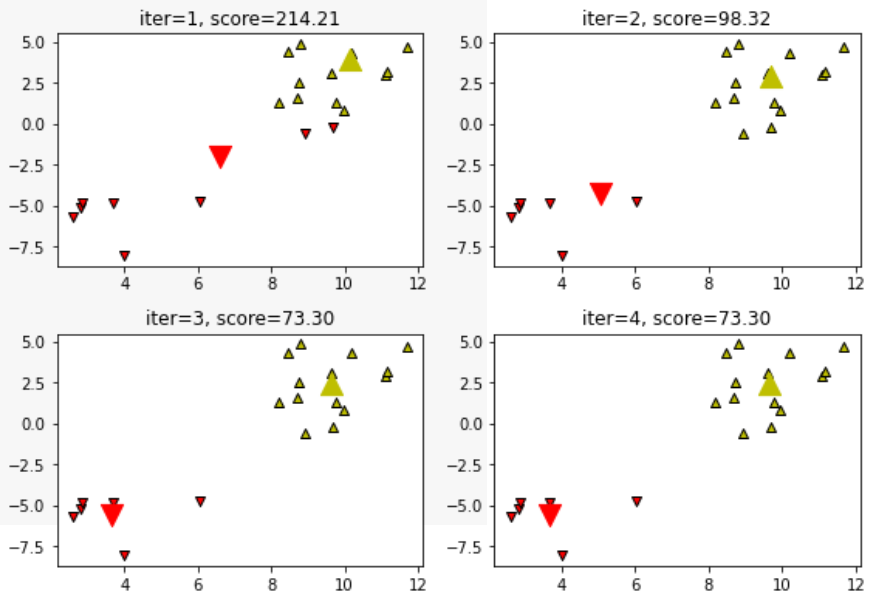
```
from sklearn.metrics import silhouette_score

for n_cluster in range(2, 11):
    kmeans = KMeans(n_clusters=n_cluster).fit(data)
    label = kmeans.labels_
    sil_coeff = silhouette_score(X, label, metric='euclidean')
    print("For n_clusters={}, The Silhouette Coefficient is {}".format(n_cluster, sil_coeff))
```

```
For n_clusters=2, The Silhouette Coefficient is 0.534100758706218
For n_clusters=3, The Silhouette Coefficient is 0.39652524043686105
For n_clusters=4, The Silhouette Coefficient is 0.3194396769494472
For n_clusters=5, The Silhouette Coefficient is 0.3649595941782594
For n_clusters=6, The Silhouette Coefficient is 0.3861406000188527
For n_clusters=7, The Silhouette Coefficient is 0.26222571689274954
For n_clusters=8, The Silhouette Coefficient is 0.28124064232185786
For n_clusters=9, The Silhouette Coefficient is 0.30740662321682494
For n_clusters=10, The Silhouette Coefficient is 0.28749926845019075
```

군집화

❖ k-평균 알고리즘(K-means algorithm)

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

X, _ = make_blobs(n_samples=20, random_state=4)

def plot_KMeans(n):
    model = KMeans(n_clusters=2, init="random", n_init=1, max_iter=n, random_state=6).fit(X)
    c0, c1 = model.cluster_centers_
    plt.scatter(X[model.labels_ == 0, 0], X[model.labels_ == 0, 1], marker='v', facecolor='r', edgecolors='k')
    plt.scatter(X[model.labels_ == 1, 0], X[model.labels_ == 1, 1], marker='^', facecolor='y', edgecolors='k')
    plt.scatter(c0[0], c0[1], marker='v', c="r", s=200)
    plt.scatter(c1[0], c1[1], marker='^', c="y", s=200)
    plt.title("iter={}, score={:5.2f}".format(n, -model.score(X)))

plt.figure(figsize=(8, 8))
plt.subplot(321)
plot_KMeans(1)
plt.subplot(322)
plot_KMeans(2)
plt.subplot(323)
plot_KMeans(3)
plt.subplot(324)
plot_KMeans(4)
plt.tight_layout()
plt.show()
```



군집화

❖ **k-평균 알고리즘**(K-means algorithm)

```
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans

# 인위적으로 2차원 데이터를 생성합니다
X, y = make_blobs(random_state=1)

# 군집 모델을 만듭니다
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=3, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=None, tol=0.0001, verbose=0)
```

군집화

❖ **k-평균 알고리즘**(K-means algorithm)

```
1 print(kmeans.labels_)
```

```
[1 2 2 2 0 0 0 2 1 1 2 2 0 1 0 0 0 1 2 2 0 2 0 1 2 0 0 1 1 0 1 1 0 1 2 0 2
 2 2 0 0 2 1 2 2 0 1 1 1 1 2 0 0 0 1 0 2 2 1 1 2 0 0 2 2 0 1 0 1 2 2 2 0 1
 1 2 0 0 1 2 1 2 2 0 1 1 1 1 2 1 0 1 1 2 2 0 0 1 0 1]
```

```
1 print(kmeans.predict(X))
```

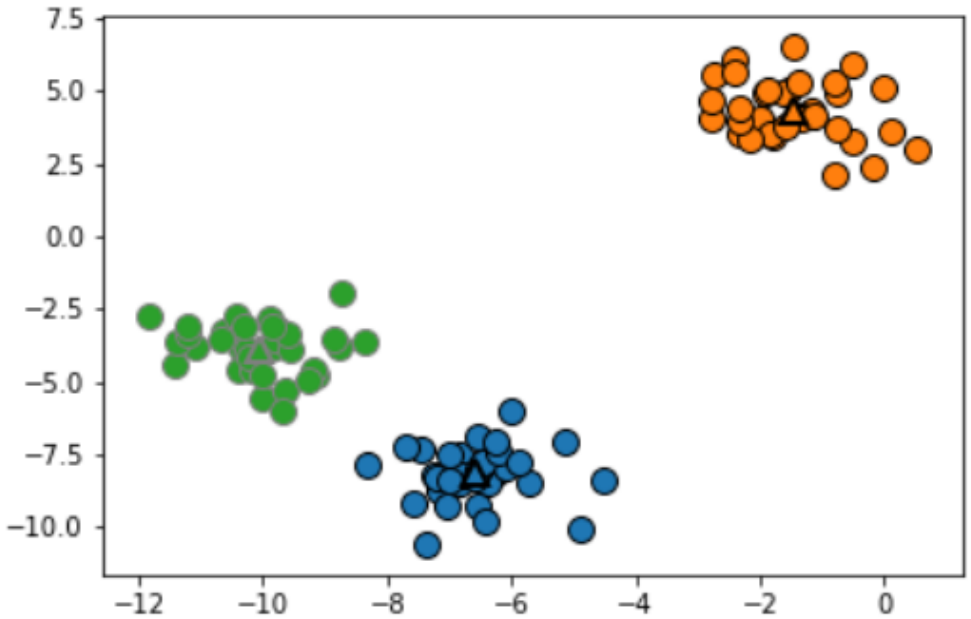
```
[1 2 2 2 0 0 0 2 1 1 2 2 0 1 0 0 0 1 2 2 0 2 0 1 2 0 0 1 1 0 1 1 0 1 2 0 2
 2 2 0 0 2 1 2 2 0 1 1 1 1 2 0 0 0 1 0 2 2 1 1 2 0 0 2 2 0 1 0 1 2 2 2 0 1
 1 2 0 0 1 2 1 2 2 0 1 1 1 1 2 1 0 1 1 2 2 0 0 1 0 1]
```

군집화

❖ **k-평균 알고리즘**(K-means algorithm)

```
1  mglearn.discrete_scatter(X[:, 0], X[:, 1], kmeans.labels_, markers='o')
2  mglearn.discrete_scatter(
3      kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], [0, 1, 2],
4      markers='^', markeredgewidth=2)
```

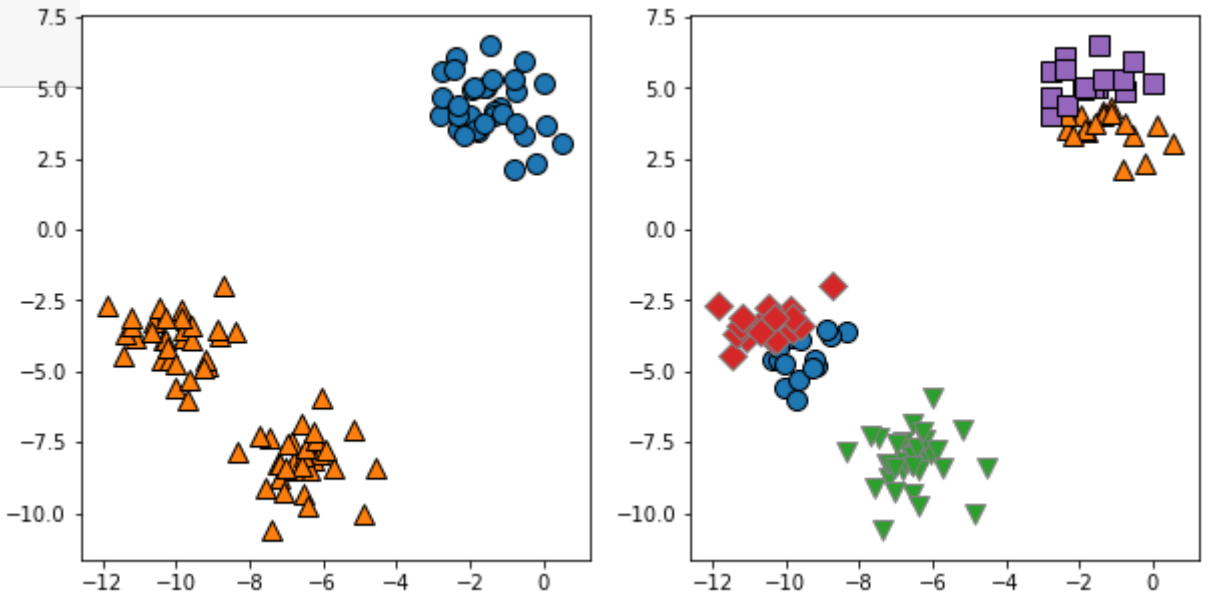
```
[<matplotlib.lines.Line2D at 0x2d5583865f8>,
 <matplotlib.lines.Line2D at 0x2d558386978>,
 <matplotlib.lines.Line2D at 0x2d558386cf8>]
```



군집화

❖ **k-평균 알고리즘**(K-means algorithm)

```
fig, axes = plt.subplots(1, 2, figsize=(10, 5))

# 두 개의 클러스터 중심을 사용합니다
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
assignments = kmeans.labels_

mglearn.discrete_scatter(X[:, 0], X[:, 1], assignments, ax=axes[0])

# 다섯 개의 클러스터 중심을 사용합니다
kmeans = KMeans(n_clusters=5)
kmeans.fit(X)
assignments = kmeans.labels_

mglearn.discrete_scatter(X[:, 0], X[:, 1], assignments, ax=axes[1])
```

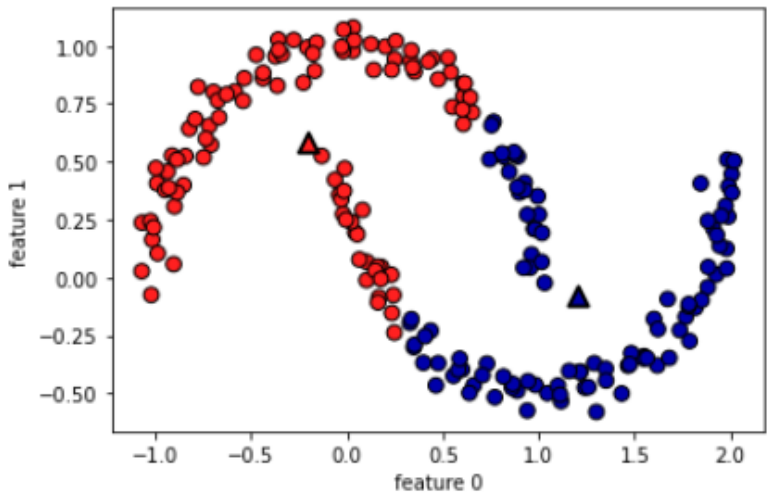


군집화

❖ k-평균 알고리즘(K-means algorithm)

```
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=200, noise=0.05, random_state=0)

# 두 개의 클러스터로 데이터에 KMeans 알고리즘을 적용합니다
kmeans = KMeans(n_clusters=2)
kmeans.fit(X)
y_pred = kmeans.predict(X)

# 클러스터 할당과 클러스터 중심을 표시합니다
plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap=mglearn.cm2, s=60, edgecolors='k')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1],
            marker='^', c=[mglearn.cm2(0), mglearn.cm2(1)], s=100, linewidth=2, edgecolors='k')
plt.xlabel("feature 0")
plt.ylabel("feature 1")
```

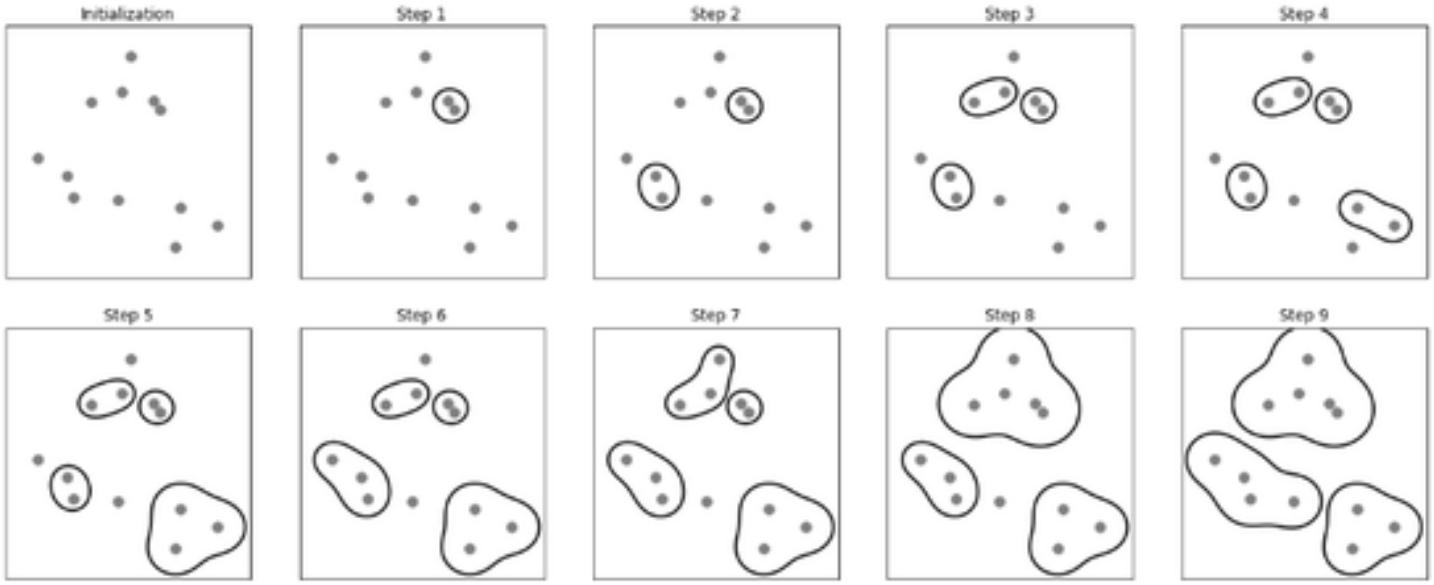
```
Text(0, 0.5, 'feature 1')
```



군집화

❖ agglomerative clustering

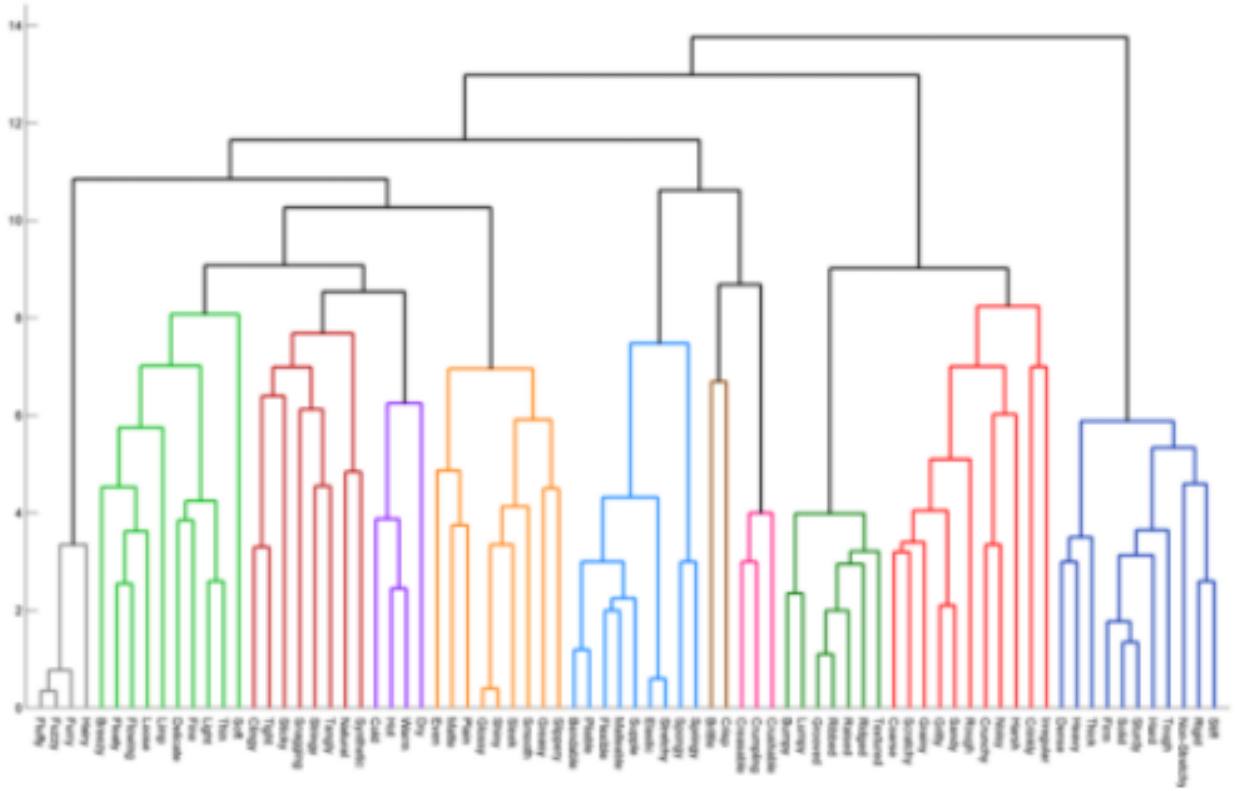
- **병합군집 알고리즘으로 각 포인트를 하나의 클러스터로 지정하고,**
  그 다음 어떤 종료 조건을 만족할 때까지 가장 비슷한 두 클러스터를 합쳐지는 방법



군집화

❖ hierarchical clustering

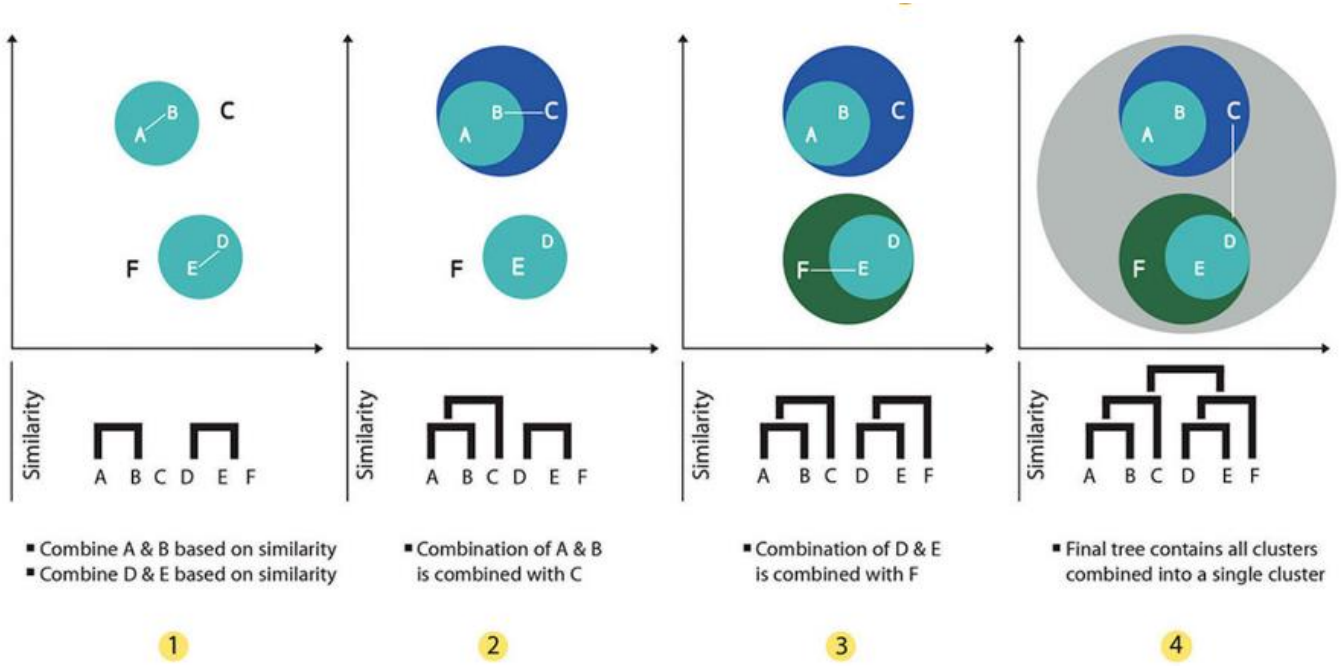
- 계층적 트리 모형을 이용해 개별 개체들을 순차적, 계층적으로 유사한 개체 내지 그룹과 통합하여 군집화를 수행하는 알고리즘



군집화

❖ hierarchical clustering

- **주어진 데이터에서 모든 두 군집간의 거리를 계산하는 알고리즘**
  - 가장 가까운 거리에 있는 데이터 묶음 / 최종적으로 하나의 클러스터로 합쳐질때까지 진행
  - Dendrogram 형태로 계층 구조 정렬
  - method : single, complete, average, centroid, ward linkage



군집화

❖ hierarchical clustering

- **주어진 데이터에서 모든 두 군집간의 거리를 계산하는 알고리즘**
  - single linkage (군집 간 element끼리의 거리 중 min을 군집 간 거리로 설정)
  - complete linkage (군집 간 element끼리의 거리 중 max를 군집 간 거리로 설정)
  - average linkage (군집 간 element끼리의 모든 거리를 average)
  - centroid (군집의 centroid끼리의 거리)
  - ward (두 군집 간 제곱합 - (군집 내 제곱합의 합))

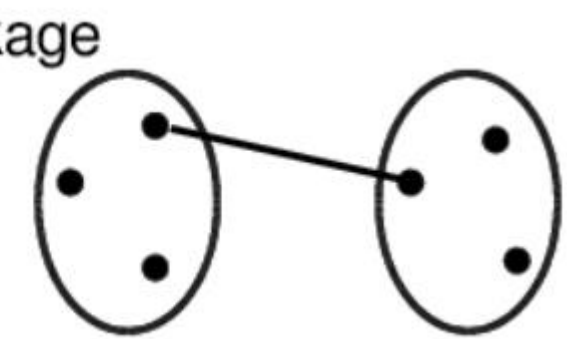
군집화

❖ hierarchical clustering

▪ 주어진 데이터에서 모든 두 군집간의 거리를 계산하는 알고리즘

- Linkage 알고리즘에 따라 클러스터 형태가 달라 적절한 방법으로 사용
- 데이터 간의 유사성이 높기 때문이며, 만일 이때 차이점이 존재한다면 데이터가 극도로 달라지기 때문에 가장 가까운 지점을 찾아 동작
- Single linkage : 두 클러스터간 가장 가까운 거리
- 군집 u의 모든 데이터 ui와 군집 v의 모든 데이터 vj의 모든 조합에 대해 데이터 사이의 거리 d(ui,vj)d(ui,vj)를 측정

$$d(u,v) = \min(d(u_i, v_j))$$

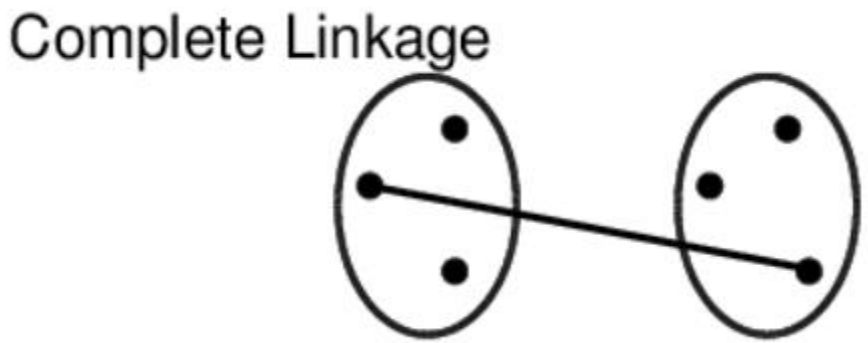


군집화

❖ hierarchical clustering

- **주어진 데이터에서 모든 두 군집간의 거리를 계산하는 알고리즘**
  - Linkage 알고리즘에 따라 클러스터 형태가 달라 적절한 방법으로 사용
  - 데이터가 클러스터 간에 멀리 떨어져 있고 클러스터 내부의 밀집도는 매우 높을 때 좋은 결과를 보여줘서.
  - Complete linkage : 두 클러스터간 가장 먼 거리
  - 군집 u의 모든 데이터 ui와 군집 v의 모든 데이터 vj의 모든 조합에 대해 데이터 사이의 거리 d(ui,vj)d(ui,vj)의 가장 큰 값

$$d(u,v) = \max(d(u_i, v_j))$$

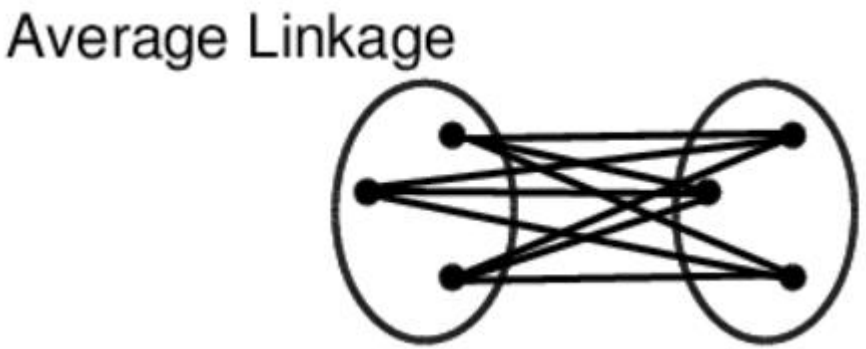


군집화

❖ hierarchical clustering

- **주어진 데이터에서 모든 두 군집간의 거리를 계산하는 알고리즘**
  - Linkage 알고리즘에 따라 클러스터 형태가 달라 적절한 방법으로 사용
  - Average linkage : 클러스터 내 모든 데이터와 , 다른 클러스터 모든 데이터 사이 거리 평균 사용
  - 군집 u의 모든 데이터 ui와 군집 v의 모든 데이터 vj의 모든 조합에 대해 데이터 사이의 거리 d(ui,vj)d(ui,vj)의 평균값

$$d(u,v) = \sum_{i,j} \frac{d(u_i, v_j)}{|u||v|}$$



군집화

❖ hierarchical clustering

▪ 주어진 데이터에서 모든 두 군집간의 거리를 계산하는 알고리즘-

- Ward
- 연결될 수 있는 군집 조합을 만들고, 군집 내 편차들의 제곱합을 기준으로 (군집의 평균과 데이터들 사이의 오차 제곱합(SSE)을 측정해) 최소 제곱합을 가지게 되는 군집끼리 연결

$$d(u,v) = \sqrt{\frac{|v|+|s|}{T}d(v,s)^2 + \frac{|v|+|t|}{T}d(v,t)^2 - \frac{|v|}{T}d(s,t)^2}$$

$$T = |v| + |s| + |t|.$$

- Centroid

$$dist(s,t) = ||c_s - c_t||_2$$

군집화

❖ hierarchical clustering

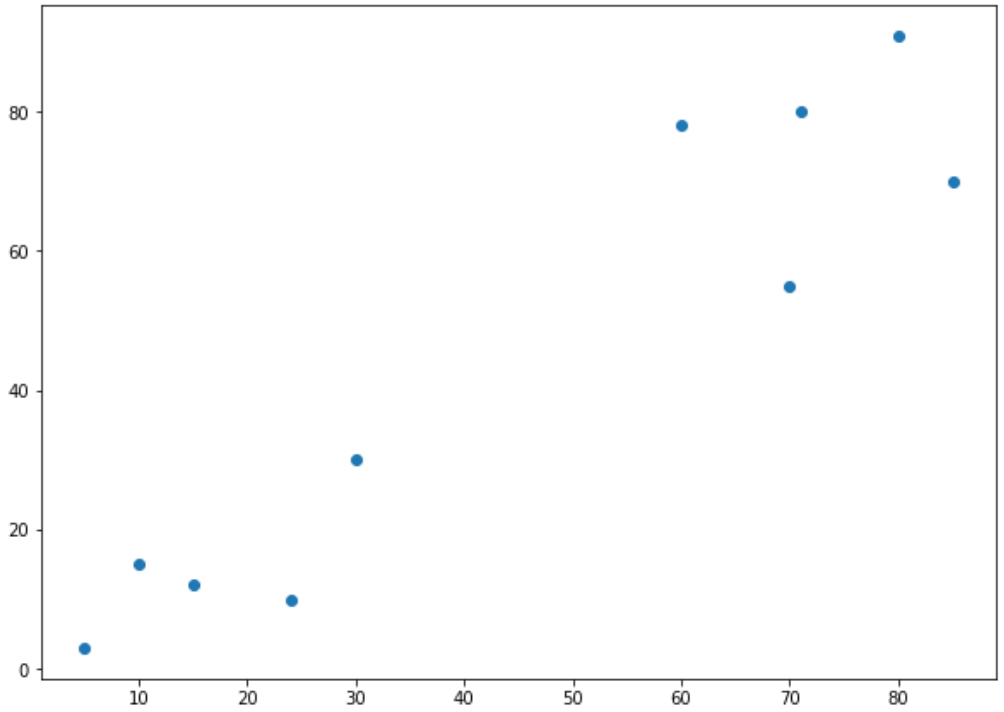
- from scipy.cluster.hierarchy import dendrogram, linkage
- from matplotlib import pyplot as plt
- ~~X = [[i] for i in [2, 8, 0, 4, 1, 9, 9, 0]] - 예시~~
- Z = linkage(X, 'ward')
- fig = plt.figure(figsize=(25, 10))
- plt.show()

군집화

❖ hierarchical clustering

```
import numpy as np

X = np.array([[5,3],
     [10,15],
     [15,12],
     [24,10],
     [30,30],
     [85,70],
     [71,80],
     [60,78],
     [70,55],
     [80,91],])
```

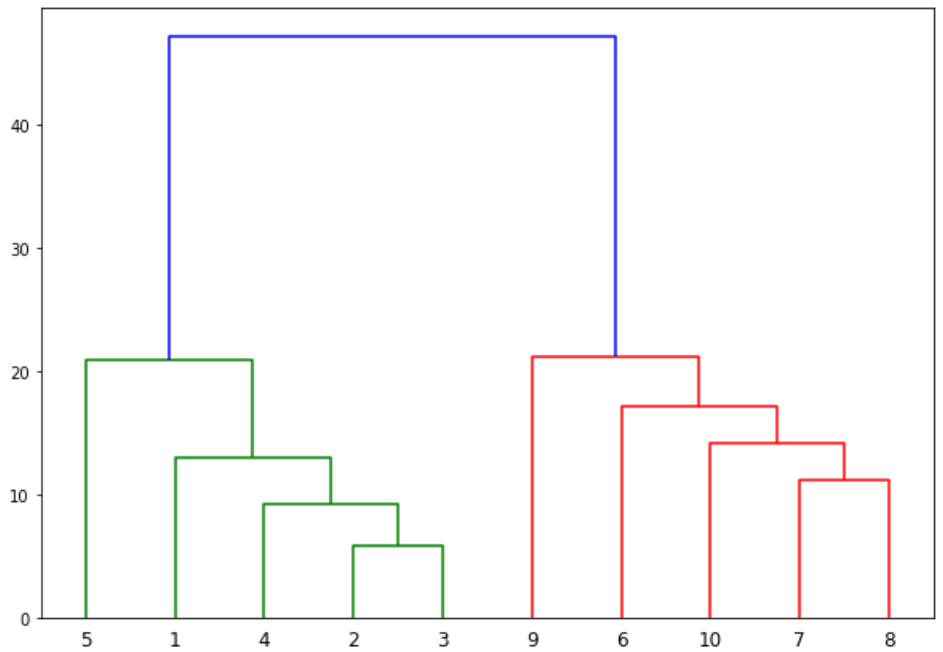
```
import matplotlib.pyplot as plt

labels = range(1, 11)
plt.figure(figsize=(10, 7))
plt.subplots_adjust(bottom=0.1)
plt.scatter(X[:,0],X[:,1])
```



군집화

❖ hierarchical clustering

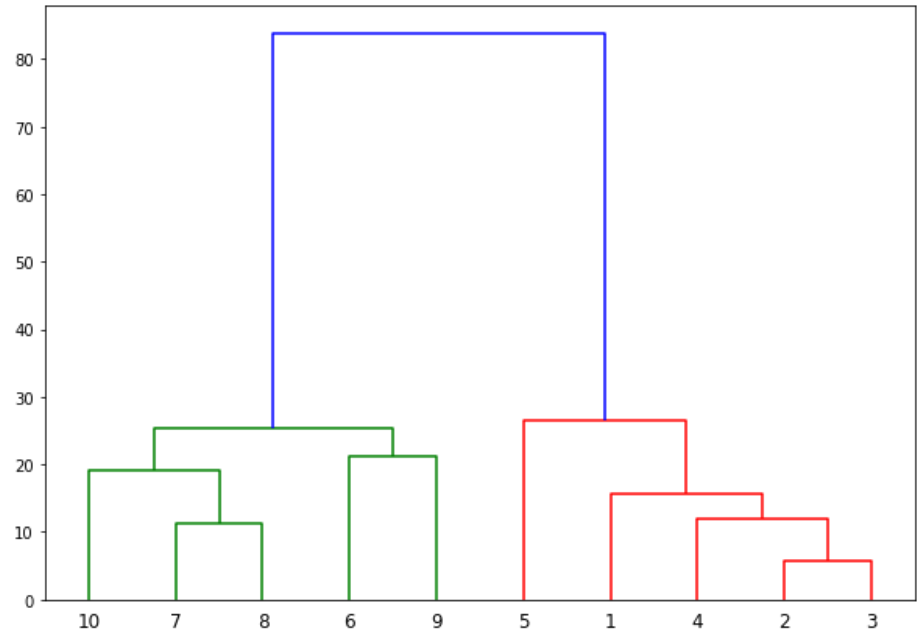
```
1  from scipy.cluster.hierarchy import dendrogram, linkage
2  from matplotlib import pyplot as plt
3
4  linked = linkage(X, 'single')
5
6  labelList = range(1, 11)
7
8  plt.figure(figsize=(10, 7))
9  dendrogram(linked,
10             orientation='top',
11             labels=labelList,
12             distance_sort='descending',
13             show_leaf_counts=True)
14 plt.show()
```



군집화

❖ hierarchical clustering

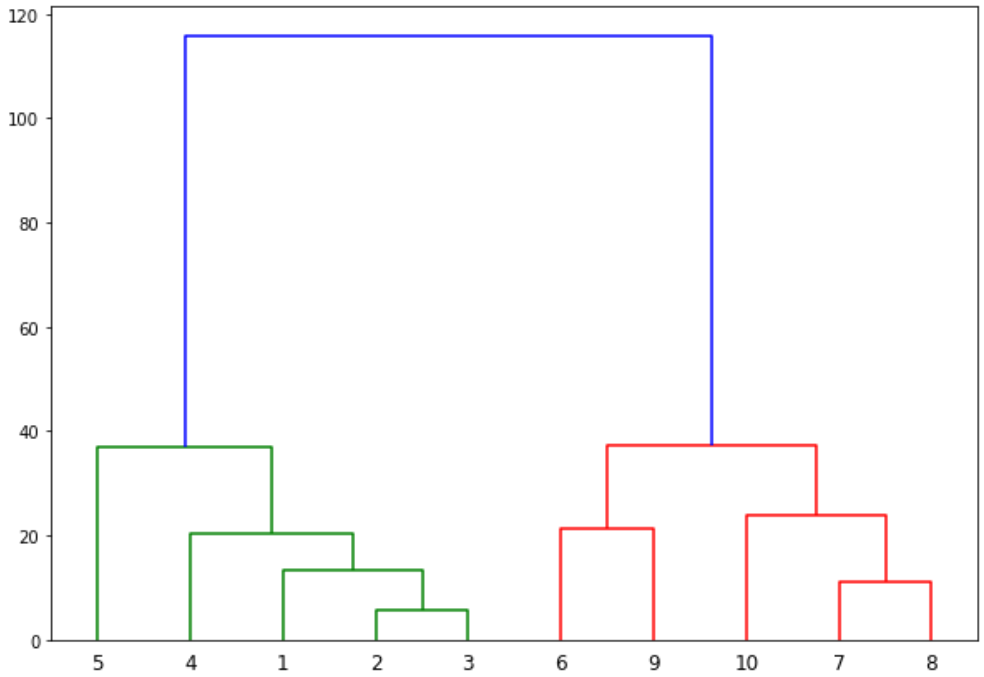
```
from scipy.cluster.hierarchy import dendrogram, linkage
from matplotlib import pyplot as plt

linked = linkage(X, 'average')

labelList = range(1, 11)

plt.figure(figsize=(10, 7))
dendrogram(linked,
           orientation='top',
           labels=labelList,
           distance_sort='descending',
           show_leaf_counts=True)
plt.show()
```



군집화

❖ hierarchical clustering

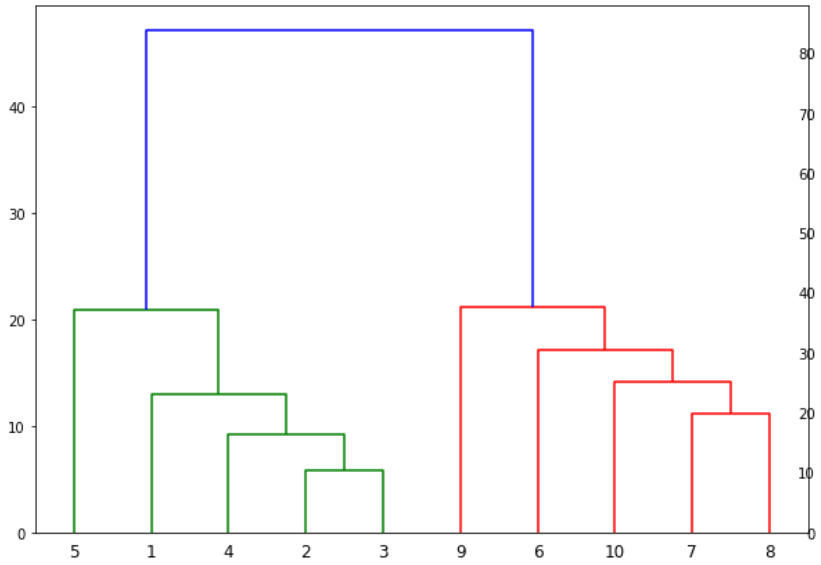
```
from scipy.cluster.hierarchy import dendrogram, linkage
from matplotlib import pyplot as plt

linked = linkage(X, 'complete')

labelList = range(1, 11)

plt.figure(figsize=(10, 7))
dendrogram(linked,
           orientation='top',
           labels=labelList,
           distance_sort='descending',
           show_leaf_counts=True)
plt.show()
```



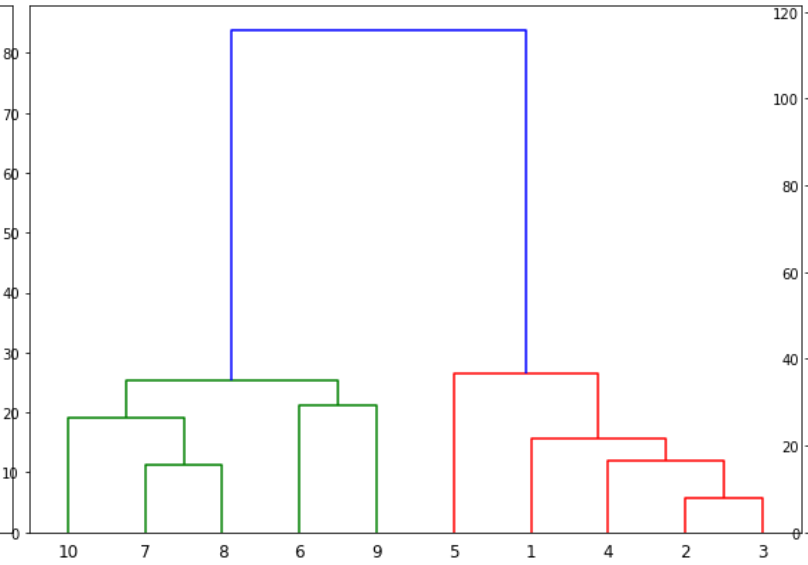
군집화

❖ **hierarchical clustering**

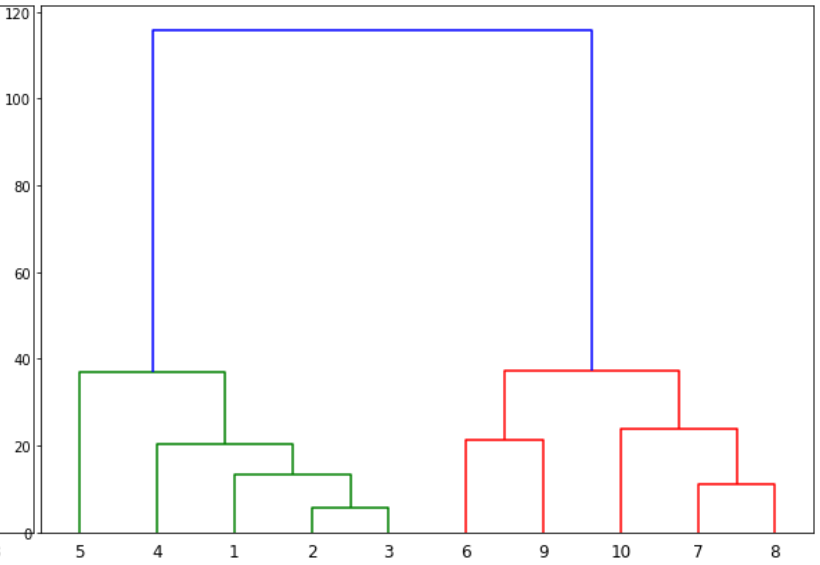
'single'



'average'



'complete'



군집화

❖ hierarchical clustering

```
1 import matplotlib.pyplot as plt
2 import pandas as pd
3 %matplotlib inline
4 import numpy as np
```

```
1 customer_data = pd.read_csv('D:/big_data/shopping_data.csv')
2 customer_data.shape
3
```

```
(200, 5)
```

```
1 customer_data.head()
2
```

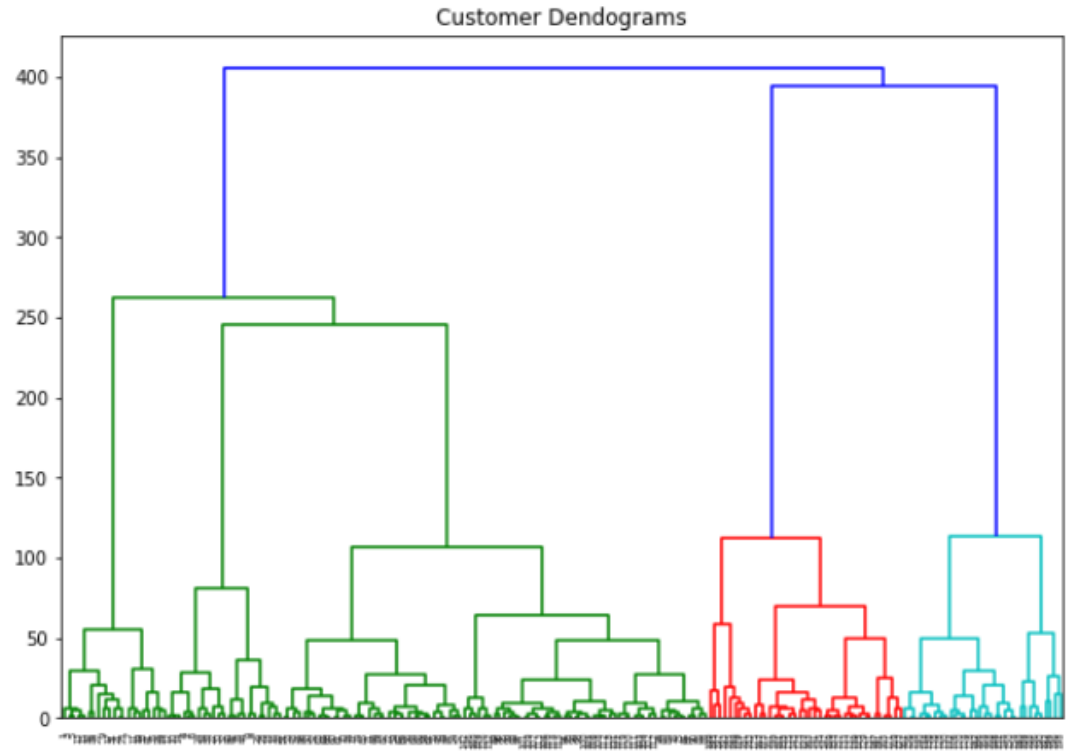
	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

군집화

❖ hierarchical clustering

```
data = customer_data.iloc[:, 3:5].values
# print(data)
```

```
import scipy.cluster.hierarchy as shc

plt.figure(figsize=(10, 7))
plt.title("Customer Dendograms")
dend = shc.dendrogram(shc.linkage(data, method='ward'))
```



군집화

❖ hierarchical clustering

```
1 from sklearn.cluster import AgglomerativeClustering
2
3 cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
4 cluster.fit_predict(data)
```

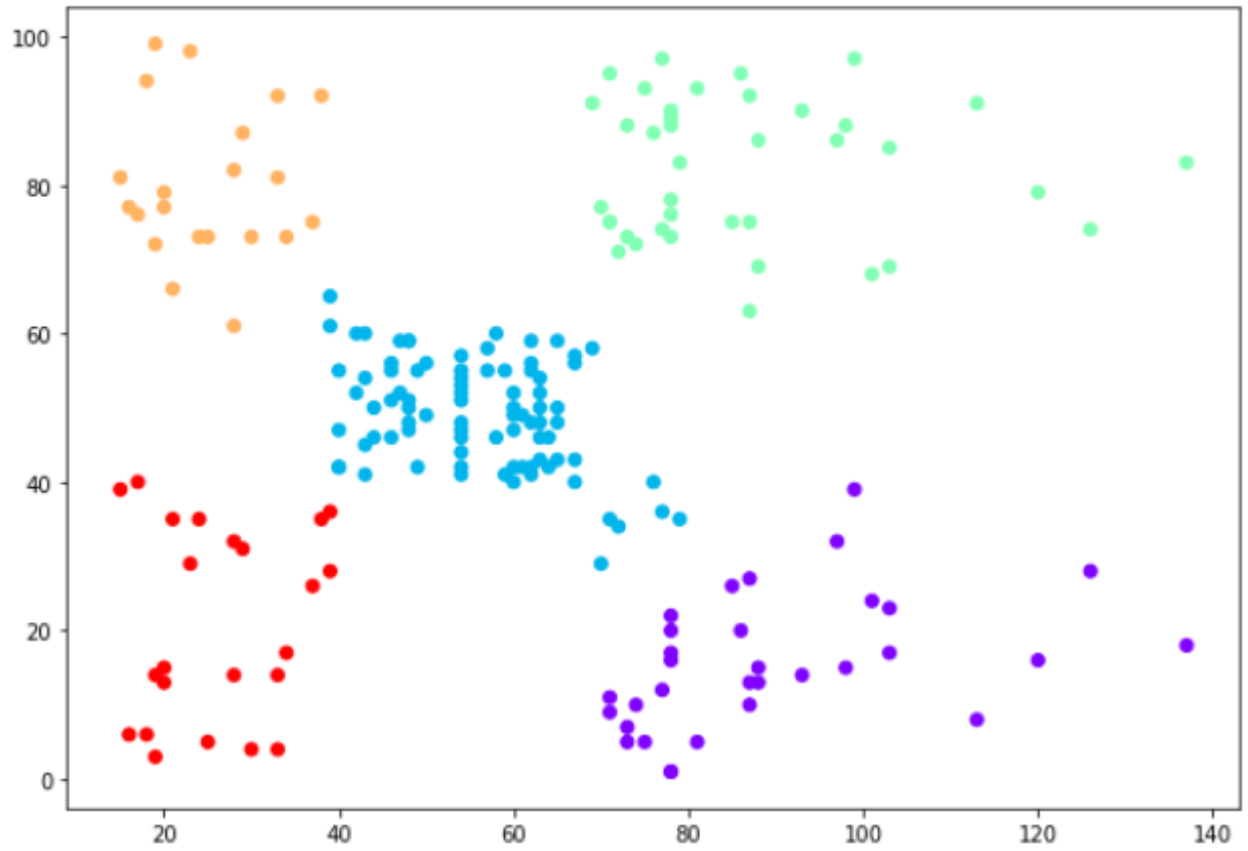
```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
       4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 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, 1, 2, 1, 2, 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2], dtype=int64)
```

군집화

❖ hierarchical clustering

```
1 plt.figure(figsize=(10, 7))
2 plt.scatter(data[:,0], data[:,1], c=cluster.labels_, cmap='rainbow')
```

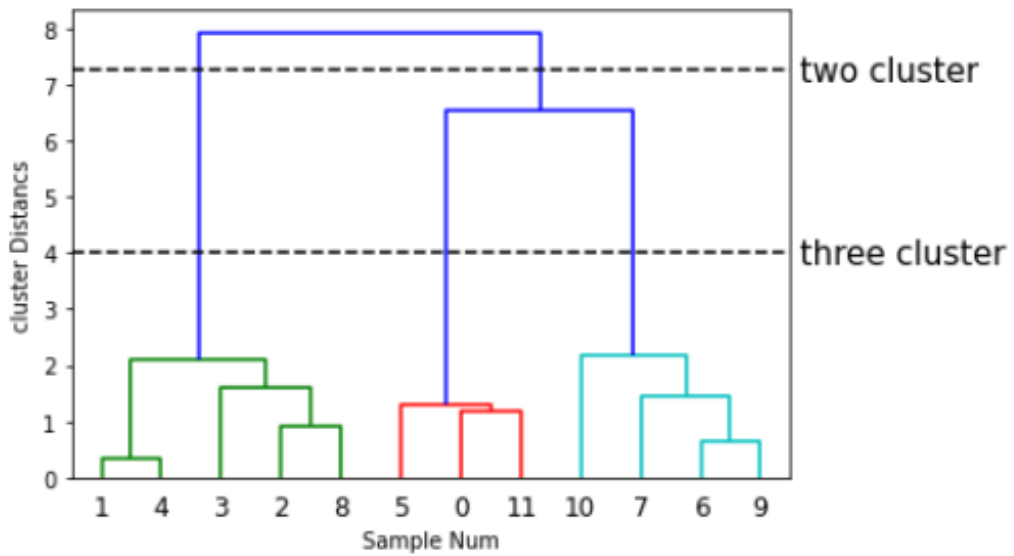
```
<matplotlib.collections.PathCollection at 0x245c6df3b70>
```



군집화

❖ hierarchical clustering

```
from scipy.cluster.hierarchy import dendrogram, ward

X, y = make_blobs(random_state=0, n_samples=12)

linkage_array = ward(X)
dendrogram(linkage_array)

ax = plt.gca()
bounds = ax.get_xbound()
ax.plot(bounds, [7.25, 7.25], '--', c='k')
ax.plot(bounds, [4, 4], '--', c='k')

ax.text(bounds[1], 7.25, ' two cluster', va='center', fontdict={'size': 15})
ax.text(bounds[1], 4, ' three cluster', va='center', fontdict={'size': 15})
plt.xlabel("Sample Num")
plt.ylabel("cluster Distancs")
```



군집화

❖ hierarchical clustering

- 임의의 기준점으로부터 사용자가 지정한 반경 거리 e(epsilon) 내에 점이 m(minimum Points)개 이상 있으면(기준점 포함) 하나의 Clustering으로 인식 (Hyper Parameter : e, m)

- 반경 거리 내에 m개의 점이 존재하는 기준점은 '중심점(Core Point)'

- 반경 거리 내에 m개의 점이 존재하지 않는 기준점이지만, 다른 중심점의 Clustering에 속하는 경우에는 '경계점(Border Point)'

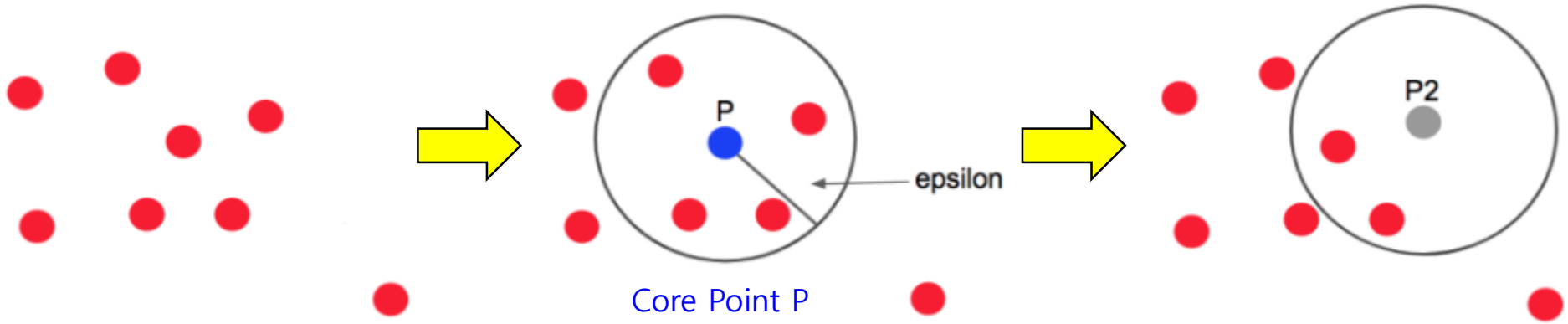
- 반경 거리 내에 m개의 점이 존재하지 않는 기준점이고, 다른 중심점의 Clustering에도 속하지 않는 경우에는 'Noise Point'

- 중심점과 중심점은 서로 연결되어 하나의 Clustering으로 묶일 수 있음

군집화

❖ DBSCAN

- **DBSCAN Clustering 수행 예시 (Hyper Parameter e 설정 & min Points = 4)**



서로 겹치지 않는
8개의 점이 존재

사용자가 지정한 epsilon에 대해
점의 개수가 min Points 이상(기준점
포함)인 경우, 기준점 P는 Core Point

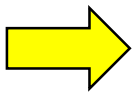
점의 개수가 min Points 미만(기준점
포함)인 경우, 기준점 P2는 Core Point가
될수 없음

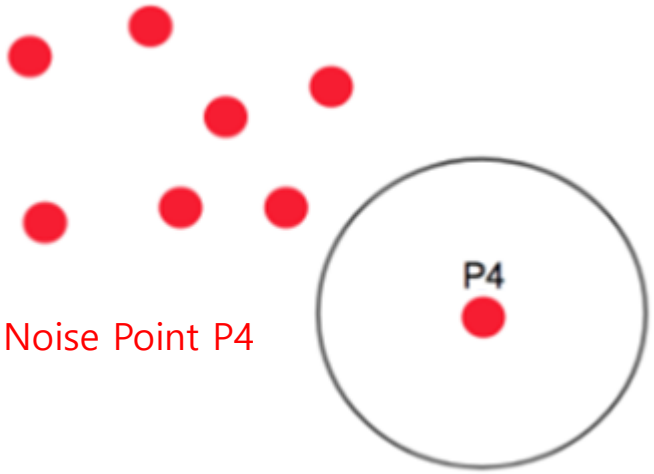
군집화

❖ DBSCAN

- **DBSCAN Clustering 수행 예시 (Hyper Parameter e 설정 & min Points = 4)**

Border Point P2





Noise Point P4

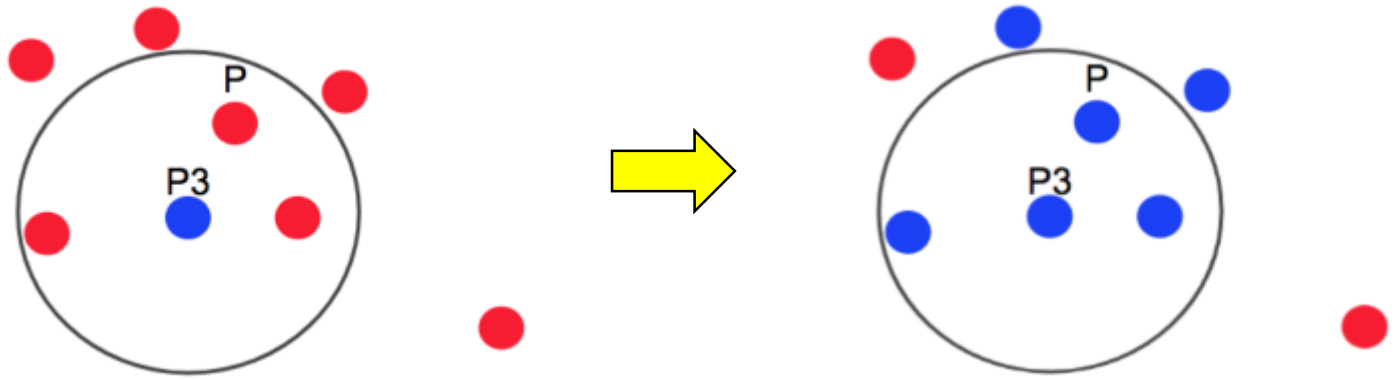
비록 점의 개수가 min Points 미만(기준점 포함)이지만,
다른 중심점의 Clustering에 속하므로
기준점 P2는 Border Point

점의 개수가 min Points 미만(기준점 포함)이고,
다른 중심점의 Clustering에 속하지 않으므로
P4는 Noise Point

군집화

❖ DBSCAN

- **DBSCAN Clustering 수행 예시 (Hyper Parameter e 설정 & min Points = 4)**



Core Point P3

Core Point P & P3

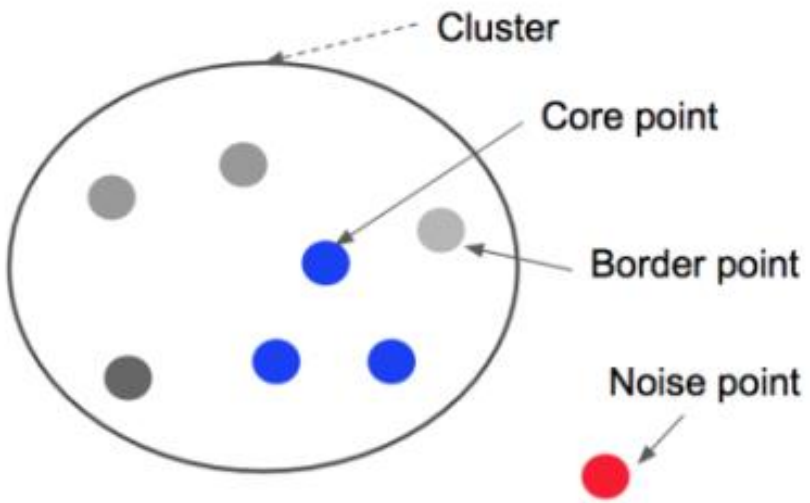
기준점 P3 역시 min Points 이상(기준점 포함)이므로 P3는 Core Point

서로 다른 Core Point에 대하여,
각 Core Point를 중심으로 하는 반경내에 다른 Core Point가 서로 포함되면 같은 Clustering으로 묶음
('Core Point끼리 연결되어 있다'고 표현)

군집화

❖ DBSCAN

- **DBSCAN Clustering 수행 예시 (Hyper Parameter e 설정 & min Points = 4)**



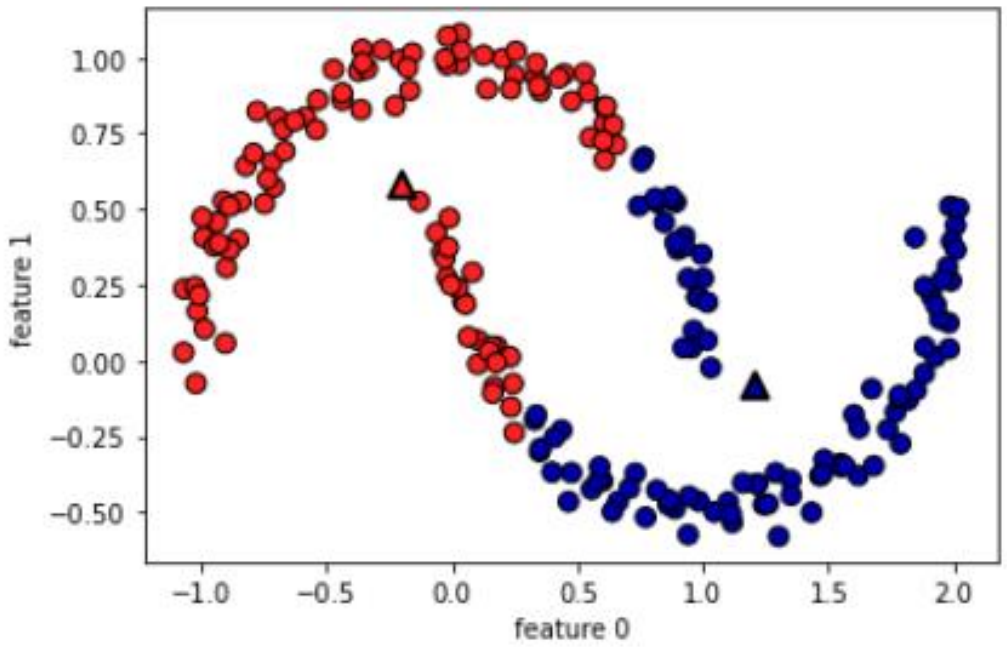
- **요약**
  - 임의의 기준점을 중심으로 epsilon 반경 내에 min Points 이상수의 점이 있으면 그 점을 중심으로 Cluster가 형성되고 그 점을 Core Point
  - 서로 다른 Core Point가 서로의 Cluster의 일부가 되면 두 Cluster를 하나의 Cluster로 연결
  - Cluster에는 속하지만 스스로 Core Point가 안되는 점을 Border Point
  - 어느 Cluster에도 속하지 않는 점을 Noise Point

군집화

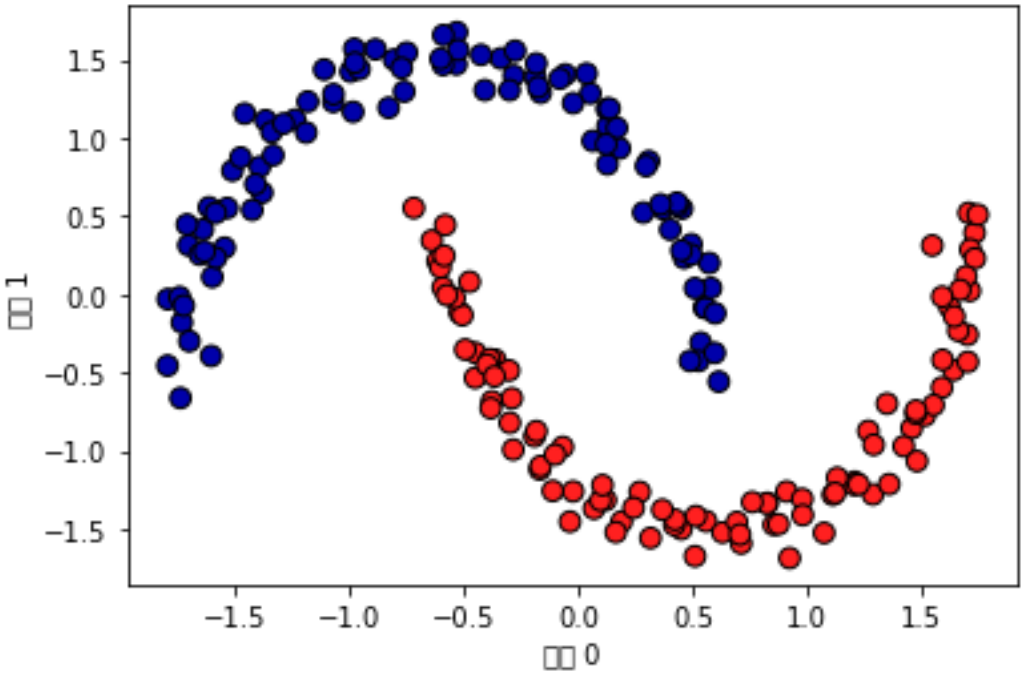
❖ DBSCAN

- **DBSCAN Clustering 수행 예시 (Hyper Parameter e 설정 & min Points = 4)**

K-means



DBSCAN



군집화

❖ DBSCAN 장, 단점

▪ 장점

- Cluster의 수를 정하지 않음
- 밀도에 따라서 Cluster를 서로 연결하기 때문에 기하학적인 모양을 갖는 Cluster도 잘 찾을 수 있음
- Noise Point를 통하여 Outlier 검출이 가능함

▪ 단점

- 계산량이 많음
- 지정 거리(epsilon) 값이 간접적으로 몇 개의 Cluster가 만들어질지 제어하므로 적절한 지정 거리를 찾으려면 모든 Feature의 Scale을 비슷한 범위로 조정해주어야 함.
- Clustering 하는 Data의 내용에 따라 지정 거리(epsilon)와 최소 샘플 개수(minimum Points) 기준이 달라짐

군집화

❖ DBSCAN 장, 단점

- from sklearn.cluster import DBSCAN
- # create model and prediction
- model = DBSCAN(eps=0.2, min_samples=6)
- predict = model.fit_predict(feature)

군집화

❖ DBSCAN

```
1 from sklearn.cluster import DBSCAN
2 import numpy as np
3 X = np.array([[1, 2], [2, 2], [2, 3],[8, 7], [8, 8], [25, 80]])
4 clustering = DBSCAN(eps=3, min_samples=2).fit(X)
5 clustering.labels_
```

```
array([ 0,  0,  0,  1,  1, -1], dtype=int64)
```

```
1
```

```
1 from sklearn.cluster import DBSCAN
2 X, y = make_blobs(random_state=0, n_samples=12)
3
4 dbscan = DBSCAN()
5 clusters = dbscan.fit_predict(X)
6 print("클러스터 레이블:₩n", clusters)
```

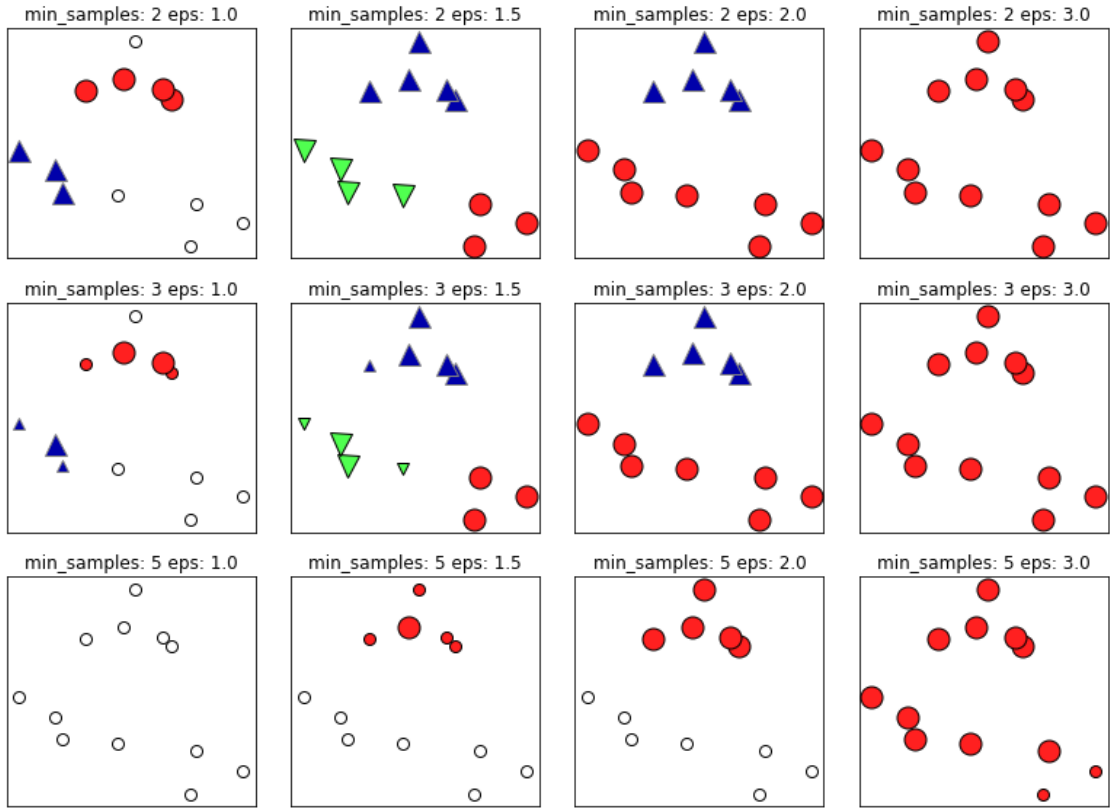
```
클러스터 레이블:
 [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
```

군집화

❖ DBSCAN

```
1 mglearn.plots.plot_dbscan()
2
```

```
min_samples: 2 eps: 1.000000  cluster: [-1  0  0 -1  0 -1  1  1  0  1 -1 -1]
min_samples: 2 eps: 1.500000  cluster: [0 1 1 1 1 0 2 2 1 2 2 0]
min_samples: 2 eps: 2.000000  cluster: [0 1 1 1 1 0 0 0 1 0 0 0]
min_samples: 2 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
min_samples: 3 eps: 1.000000  cluster: [-1  0  0 -1  0 -1  1  1  0  1 -1 -1]
min_samples: 3 eps: 1.500000  cluster: [0 1 1 1 1 0 2 2 1 2 2 0]
min_samples: 3 eps: 2.000000  cluster: [0 1 1 1 1 0 0 0 1 0 0 0]
min_samples: 3 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
min_samples: 5 eps: 1.000000  cluster: [-1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1 -1]
min_samples: 5 eps: 1.500000  cluster: [-1  0  0  0  0 -1 -1 -1  0 -1 -1 -1]
min_samples: 5 eps: 2.000000  cluster: [-1  0  0  0  0 -1 -1 -1  0 -1 -1 -1]
min_samples: 5 eps: 3.000000  cluster: [0 0 0 0 0 0 0 0 0 0 0 0]
```



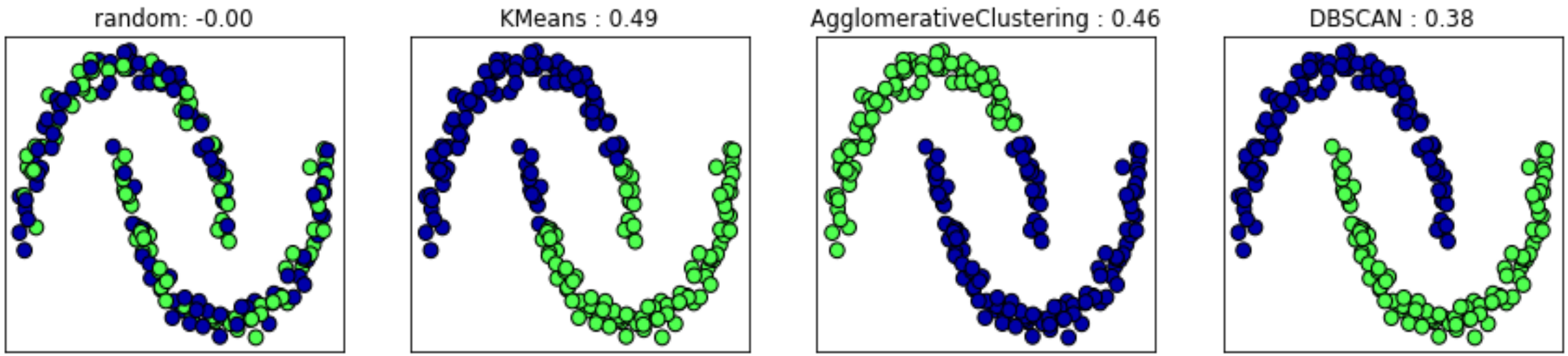
군집화

❖ DBSCAN

```
from sklearn.metrics.cluster import silhouette_score

X, y = make_moons(n_samples=200, noise=0.05, random_state=0)

# 평균이 0, 분산이 1이 되도록 데이터의 스케일을 조정합니다
scaler = StandardScaler()
scaler.fit(X)
X_scaled = scaler.transform(X)

fig, axes = plt.subplots(1, 4, figsize=(15, 3),
                         subplot_kw={'xticks': (), 'yticks': ()})

# 비교를 위해 무작위로 클러스터 할당을 합니다
random_state = np.random.RandomState(seed=0)
random_clusters = random_state.randint(low=0, high=2, size=len(X))

# 무작위 할당한 클러스터를 그립니다
axes[0].scatter(X_scaled[:, 0], X_scaled[:, 1], c=random_clusters,
                cmap=mglearn.cm3, s=60, edgecolors='black')
axes[0].set_title("random: {:.2f}".format(
        silhouette_score(X_scaled, random_clusters)))

algorithms = [KMeans(n_clusters=2), AgglomerativeClustering(n_clusters=2),
              DBSCAN()]

for ax, algorithm in zip(axes[1:], algorithms):
    clusters = algorithm.fit_predict(X_scaled)
    # 클러스터 할당과 클러스터 중심을 그립니다
    ax.scatter(X_scaled[:, 0], X_scaled[:, 1], c=clusters, cmap=mglearn.cm3,
               s=60, edgecolors='black')
    ax.set_title("{} : {:.2f}".format(algorithm.__class__.__name__,
                                      silhouette_score(X_scaled, clusters)))
```

군집화

❖ **DBSCAN**


random: -0.00
KMeans : 0.49
AgglomerativeClustering : 0.46
DBSCAN : 0.38

군집화

❖ DBSCAN

```
import pandas as pd
iris = datasets.load_iris()

labels = pd.DataFrame(iris.target)
labels.columns=['labels']
data = pd.DataFrame(iris.data)
data.columns=['Sepal length','Sepal width','Petal length','Petal width']
data = pd.concat([data,labels],axis=1)

data.head()

```

	Sepal length	Sepal width	Petal length	Petal width	labels
0	5.1	3.5	1.4	0.2	0
1	4.9	3.0	1.4	0.2	0
2	4.7	3.2	1.3	0.2	0
3	4.6	3.1	1.5	0.2	0
4	5.0	3.6	1.4	0.2	0

군집화

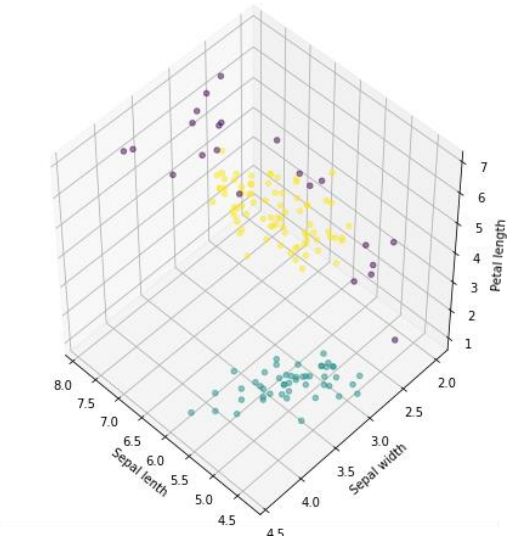
❖ DBSCAN

```
from sklearn.cluster import DBSCAN
import matplotlib.pyplot  as plt
import seaborn as sns

# create model and prediction
model = DBSCAN(min_samples=6)
predict = pd.DataFrame(model.fit_predict(feature))
predict.columns=['predict']

# concatenate labels to df as a new column
r = pd.concat([feature,predict],axis=1)

```

```
ct = pd.crosstab(data['labels'],r['predict'])
print (ct)


```

predict	-1	0	1
labels			
0	1	49	0
1	6	0	44
2	15	0	35

```
from mpl_toolkits.mplot3d import Axes3D
# scatter plot
fig = plt.figure( figsize=(6,6))
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
ax.scatter(r['Sepal length'],r['Sepal width'],r['Petal length'],c=r['predict'],alpha=0.5)
ax.set_xlabel('Sepal lenth')
ax.set_ylabel('Sepal width')
ax.set_zlabel('Petal length')
plt.show()

```



reference

- 모든 강의자료는 고려대 강필성 교수님 강의와 김성범 교수님 강의를 참고했음
- ratsgo’s blog ,https://ratsgo.github.io/
- https://leonard92.tistory.com/9
- https://leedakyeong.tistory.com/entry/%EA%B5%B0%EC%A7%91%EA%B3%BC-%EB%B6%84%EB%A5%98%EC%9D%98-%EC%B0%A8%EC%9D%B4-difference-of-clustering-and-classification
- http://hleecaster.com/ml-kmeans-clustering-concept/
- https://astralworld58.tistory.com/84
- https://ratsgo.github.io/machine%20learning/2017/04/19/KC/
- https://msmskim.tistory.com/66
- http://bcho.tistory.com/1205
- 안드레아스 뮐러, 세라 가이도 지음, 박해선 옮김, "파이썬 라이브러리를 활용한 머신러닝", 한빛미디어(2017)

감사합니다