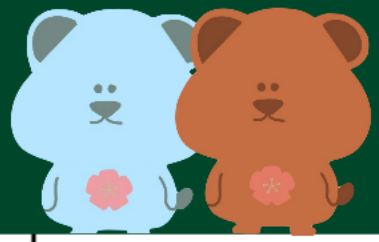
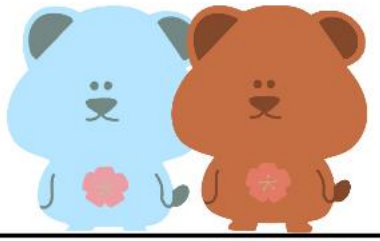


10주차 발표

DA팀 김예진 오수진 이의진

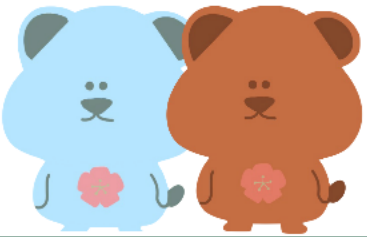
EWHA EURON

목차



EWHA EURON

01. 군집화



EWHA EURON

1.1 지도학습 vs. 비지도학습

지도학습(Supervised Learning) vs. 비지도학습(Un-supervised Learning)

지도학습(Supervised Learning)	비지도학습(Un-supervised Learning)
정답이 있는(레이블 된) 데이터를 활용해 데이터를 학습시키는 것	정답 레이블이 없는 데이터를 비슷한 특징끼리 군집화하여 새로운 데이터에 대한 결과를 예측하는 것
- 분류 - 회귀 - 추천 시스템	- 군집화(클러스터링) - 차원 축소 - 연관 규칙 학습

1.2 군집화 개념

군집화(Clustering)

군집화(clustering)

: 주어진 데이터 집합을 유사한 데이터들의 그룹으로 나누는 것.(이렇게 나누어진 유사한 데이터의 그룹을 군집이라고 함)
즉, 데이터 내에 숨어있는 패턴, 그룹을 파악하여 서로 묶는 것

군집화 목표:

- 군집 간 분산 최대화
- 군집 내 분산 최소화

➔ 즉, 동일한 군집에 속한 데이터들을 서로 유사하게 그룹핑하는 것(다른 군집에 속한 데이터들은 서로 다를수록 좋음)

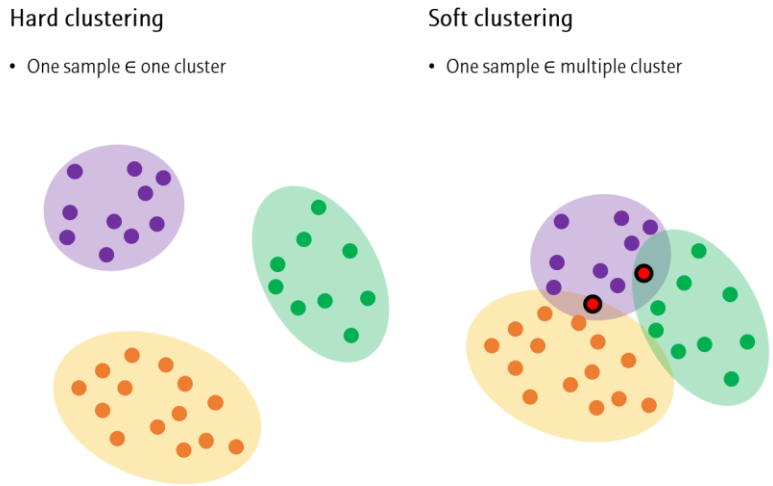
분류 vs 군집화

- 분류: 지도 학습 방식. 라벨이 있을 때, 새로운 데이터의 그룹을 예측하기 위한 목적으로 하는 분석기법
- 군집화: 비지도 학습 방식. 각 개체의 범주가 군집의 정보를 모를 때(즉 라벨이 없을 때),
  데이터 자체의 특성에 대해 알고자 하는 목적
  (군집화는 데이터 내에 숨어있는 별도의 그룹을 찾아서 의미 부여, 동일한 분류 값에 속하더라도
  그 안에서 더 세분화된 군집화 추구, 서로 다른 분류 값의 데이터도 더 넓은 군집화 레벨화 등의 영역 가짐)

1.3 군집화 알고리즘

군집화 알고리즘

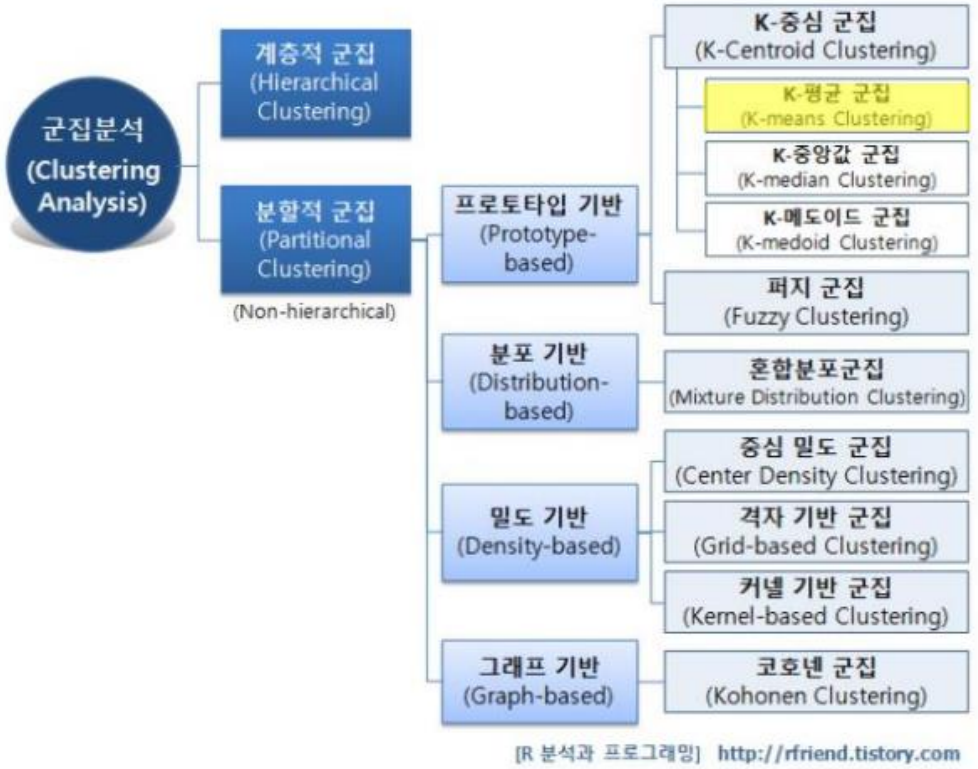
hard clustering	soft clustering
한 데이터가 정확히 하나의 군집에 속하는 것	한 데이터가 다수의 군집에 속하는 것
- hierarchical clustering - k-means - dbscan - optics	- topic models - fcm - soft k-means



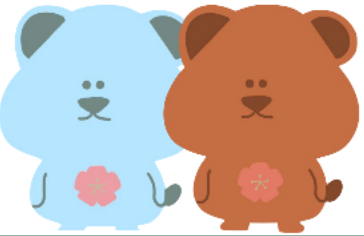
군집화 알고리즘
K-평균 군집화(K-means Clustering)
평균이동
디비스캔 군집화(DBSCAN Clustering)
가우스 혼합 모델 GMM(Gaussian Mixture Model)
병합군집
베이즈가우시안 혼합 모델

1.3 군집화 알고리즘

군집화 알고리즘(참고용)



02. K-means Clustering



EWHA EURON

2.1 k-평균 군집화

k-평균 군집화

군집화(Clustering)에서 가장 일반적으로 사용되는 알고리즘으로,
각 군집의 평균(mean)을 활용하여 K개의 군집으로 묶는 기법
➔ 군집 중심점이라는 특정한 임의의 지점을 선택해 해당 중심에 가장 가까운 포인트들을 선택하는 군집화 기법

1) 군집의 개수(K) 설정

2) 초기 중심점 설정

3) 데이터를 군집에 할당(배정)

4) 중심점 재설정(갱신)

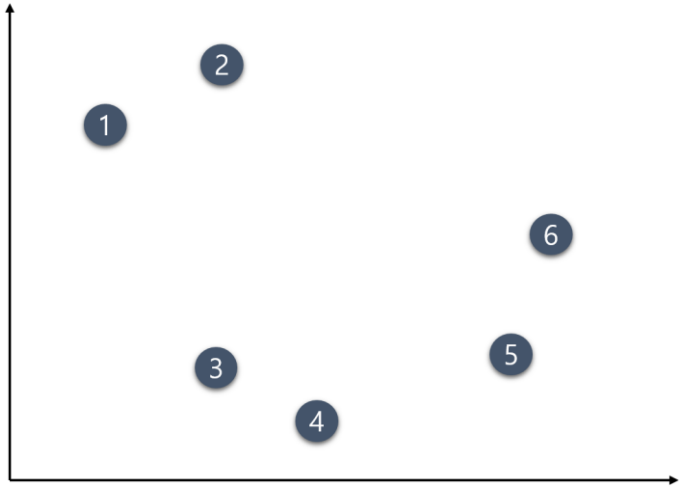
5) 데이터를 군집에 재할당(배정)

중심점의 위치가 더이상
변하지 않을 때까지 반복

2.1 k-평균 군집화

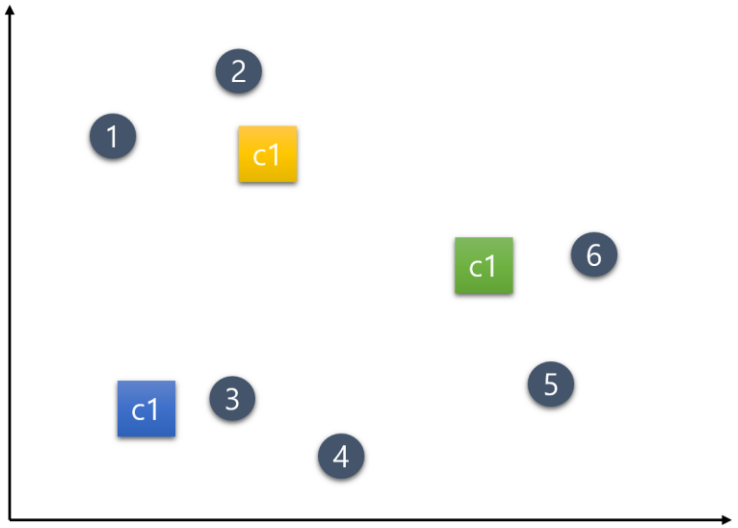
k-평균 군집화 원리(과정)

1. **군집의 개수(k) 설정**
   몇 개의 군집으로 군집화 할 지 사람이 결정.
   (k-평균의 한계점:
   군집 개수 설정에 따라 결과가 크게 달라짐)
   군집 개수 설정 방법론 1) Rule of thumb
   2) Elbow Method 3) 정보 기준 접근법



k는 임의대로 3이라고 설정

2. **초기 중심점 설정**
   k개의 군집 중심점(center of cluster, centroid) 설정.
   군집 중심점으로 어떤 값을 선택하는가에 따라
   성능이 크게 달라짐.
   초기 중심값 설정 방법론 1) Randomly select
   2) Manually assign 3) K-means++(k-means에서 사용됨)



노랑 c1 = c1
파랑 c1 = c2
초록 c1 = c3

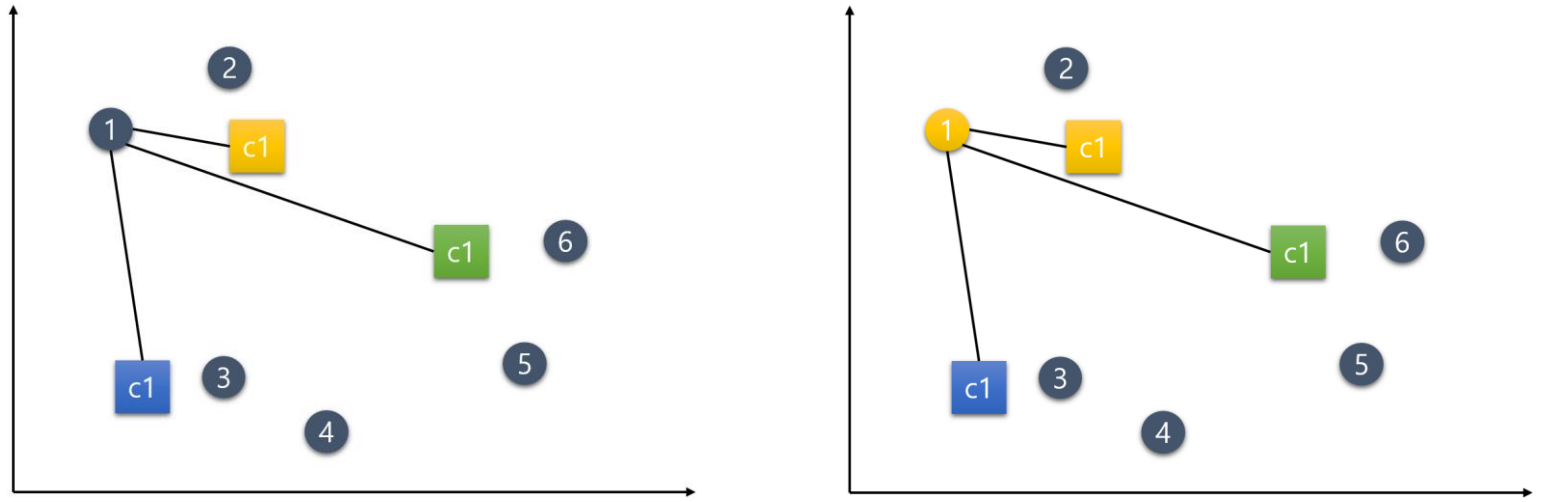
여기서는 편의상 초기 중심점(Centroid)
c1, c2, c3는 위와 같이 랜덤으로 설정

2.1 k-평균 군집화

k-평균 군집화 원리(과정)

3. 데이터를 군집에 할당(배정)

거리 상 가장 가까운 군집(중심점)으로 주어진 모든 데이터를 할당 또는 배정함
(거리 측정 방법은 일반적으로 유클리드 거리로 측정)

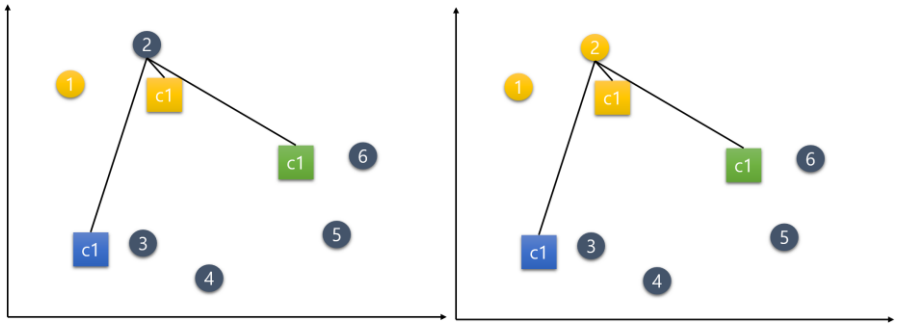


노랑 c1 = c1
파랑 c1 = c2
초록 c1 = c3

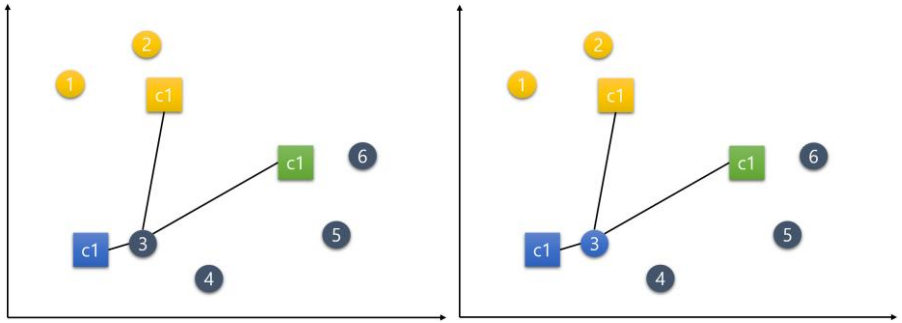
1번 데이터부터 6번 데이터까지 각각의 중심점에 가까운 군집으로 배정한다.
- 위의 그림과 같이 c1, c2, c3 중심점으로부터 1번 데이터의 유클리드 거리를 측정한다.
- 가장 가까이 위치하는 중심점의 군집으로 배정한다.
- 1번 데이터의 경우 c1 중심점과 가장 가까우므로 노란색으로 바뀐다.
- 1번 외 나머지 데이터들도 같은 방식으로 배정한다.

2.1 k-평균 군집화

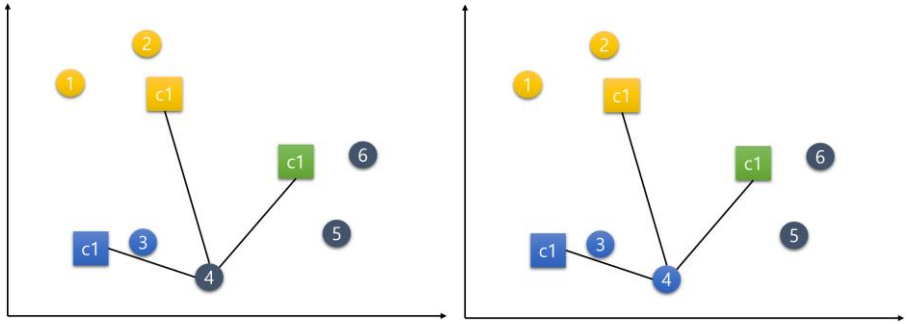
3. 데이터를 군집에 할당(배정)



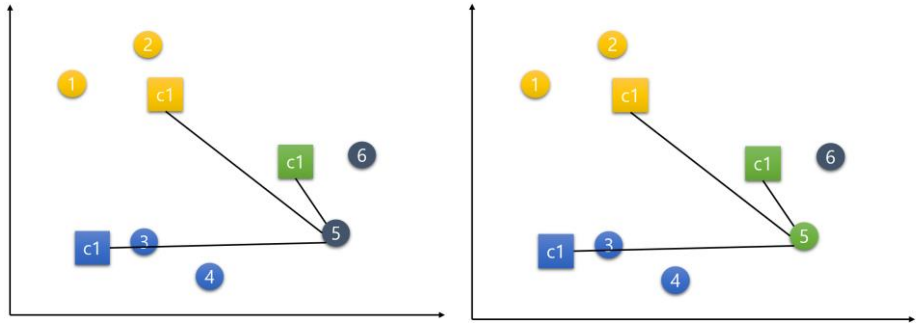
2번 데이터



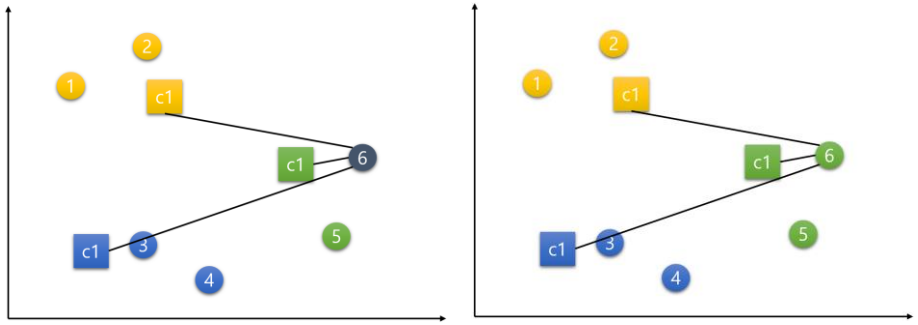
3번 데이터



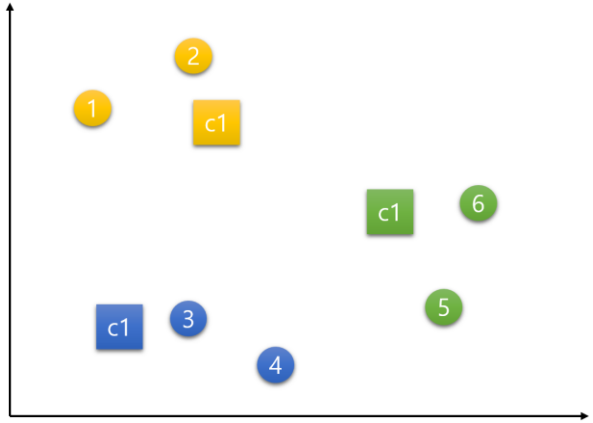
4번 데이터



5번 데이터



6번 데이터



➔ 모든 데이터들이 한 번씩 각각의 군집들로 배정됨.

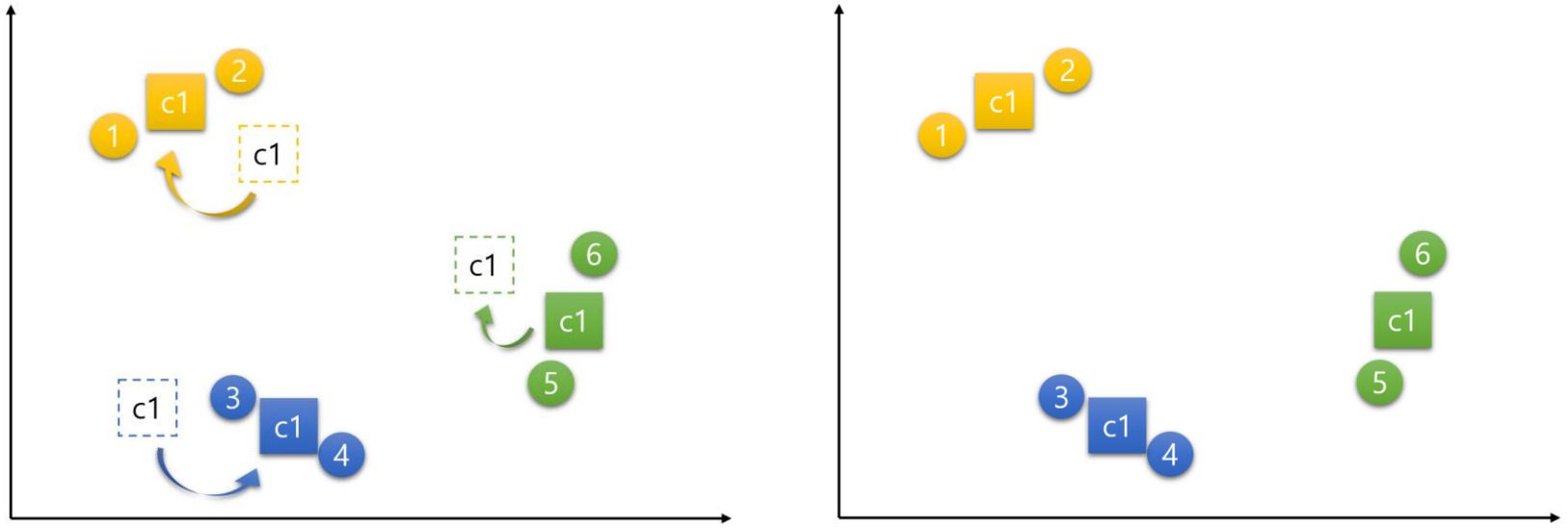
2.1 k-평균 군집화

k-평균 군집화 원리(과정)

4. 중심점 재설정(갱신)

모든 주어진 데이터의 군집 배정이 끝나면,
군집의 중심점을 그 군집에 속하는 데이터들의 가장 중간(평균)에 위치한 지점으로 재설정함.

c1, c2, c3 각각의 중심점은 그 군집의 속하는 데이터들의 가장 중간(평균)에 위치한 지점으로 재설정함.
중심점 c1은 데이터 1, 2의 평균인 지점으로, 중심점 c2는 데이터 3, 4의 평균인 지점으로,
중심점 c3는 데이터 5, 6의 평균인 지점으로 갱신됨.

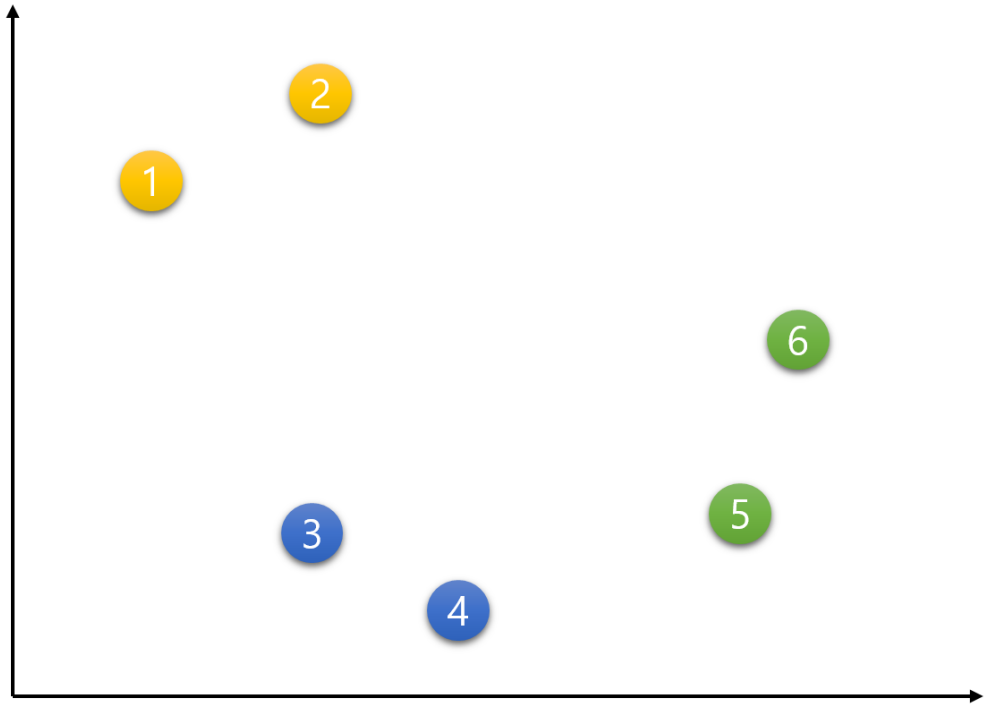


2.1 k-평균 군집화

k-평균 군집화 원리(과정)

5. 데이터를 군집에 재할당(배정)
   step 3에서 했던 방법과 똑같이 시행하며, 더 이상 중심점의 이동이 없을 때까지 step 4와 step 5를 반복

   더 이상 중심점의 이동이 없을 때까지 step 4와 step 5를 반복한다고 했는데 이 예시는 더 이상 이동이 없어 종료됨.
   즉, 아래 그림이 군집화된 최종 결과이다.



2.2 k-평균 알고리즘 장단점

k-평균 알고리즘 장단점

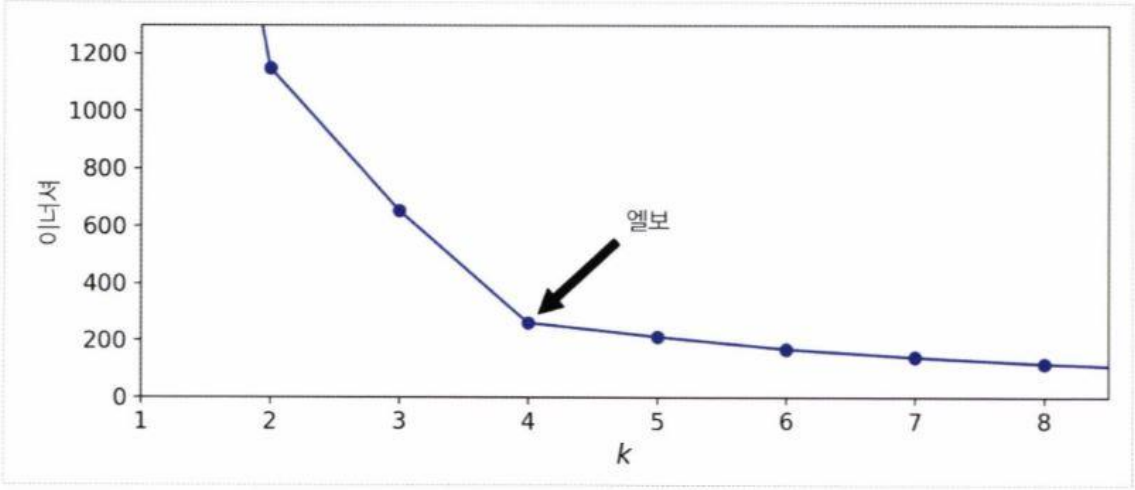
장점	단점
일반적인 군집화에서 가장 많이 활용되는 알고리즘	거리 기반 알고리즘으로 속성의 개수가 매우 많을 경우 군집화 정확도가 떨어짐. (이를 위해 PCA로 차원 감소를 적용해야 할 수도 있음)
알고리즘이 쉽고 간결함	반복을 수행하는데, 반복 횟수가 많을 경우 수행 시간이 매우 느려짐
데이터에 대한 사전 정보가 필요하지 않음	몇 개의 군집을 선택해야 할지 가이드하기가 어려움
	클러스터의 크기나 밀집도가 서로 다르거나 원형의 형태가 아닌 경우 결과가 좋지 못함 (즉, 데이터셋에 따라 잘 수행할 수 있는 군집 알고리즘이 다름. 타원형 데이터셋에 대해서는 적절하지 않음.)

2.3 군집 평가 지표

이너셔와 엘보우 → 최적의 클러스터 개수 찾기

이너셔: 클러스터 중심과 클러스터에 속한 샘플 사이의 거리를 잴 수 있는데, 이 거리의 제곱 합을 이너셔라고 함.
- 클러스터에 속한 샘플이 얼마나 가깝게 모여 있는지를 나타내는 값
- 클러스터 개수와 이너셔는 반비례 관계
  (일반적으로 클러스터 개수가 늘어나면 각각의 클러스터 크기는 줄어들어, 이너셔도 함께 줄어들기 때문)

엘보우: 클러스터 개수를 증가시켜서 이너셔를 그래프로 그렸을 때, 감소하는 속도가 꺾이는 지점(=최적의 클러스터 개수)
- 이 지점부터는 클러스터 개수를 늘려도 클러스터에 잘 밀집된 정도가 크게 개선되지 않음
  (= 이너셔가 크게 줄어들지 않음)



2.3 군집 평가 지표

실루엣 분석

실루엣 분석은 각 군집 간의 거리가 얼마나 효율적으로 분리돼 있는지를 나타내며, 실루엣 계수를 기반으로 함.

효율적 분리?

다른 군집과의 거리는 떨어져 있고, 동일 군집끼리의 데이터는 서로 가깝게 잘 뭉쳐 있는 것을 의미함.
=> 군집화가 잘 될수록 개별 군집은 비슷한 정도의 여유 공간을 가지고 떨어져있음.

실루엣 계수?

개별 데이터가 가지는 군집화 지표, 해당 데이터가 같은 군집 내의 데이터와 얼마나 가깝게 군집화 돼 있고,
다른 군집에 있는 데이터와는 얼마나 멀리 분리돼 있는지를 나타내는 지표.

$$S_i = \frac{b_i - a_i}{Max(a_i, b_i)}$$

- si: 실루엣 계수
- ai: 해당 데이터 포인트와 같은 군집 내에 있는 다른 데이터 포인트와의 거리를 평균한 값
- bi: 해당 데이터 포인트가 속하지 않은 군집 중 가장 가까운 군집과의 평균 거리
- 두 군집간의 거리가 얼마나 떨어져 있는가의 값인 b(i)-a(i)를 정규화하기 위해 Max(ai, bi)로 나눠줌

➔ 실루엣 계수는 -1과 1 사이의 값을 가짐
- 1에 가까울 수록 근처의 군집과 멀리 떨어져 있음
- 0에 가까울수록 근처 군집과 가까워짐
- 음수 값은 아예 다른 군집에 데이터 포인트가 할당됐음을 의미함

좋은 군집화의 기준
1. 전체 실루엣 계수의 평균값이 0~1 사이의 값을 가지며, 1에 가까울 수록 좋음
2. 전체 실루엣 계수의 평균값과 더불어 개별 군집의 평균값의 편차가 크지 않아야 함. 즉, 개별 군집의 실루엣 계수의 평균값이 전체 실루엣 계수의 평균값에서 크게 벗어나지 않는 것이 중요함.

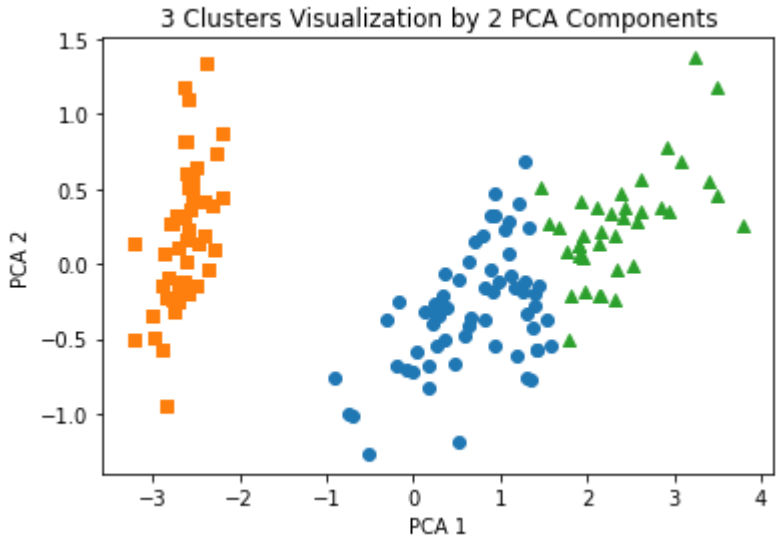
2.4 사이킷런을 이용한 k-평균 군집화

iris 데이터셋 k-평균 군집화

```
iris = load_iris()
# DataFrame으로 변환
irisDF = pd.DataFrame(data=iris.data, columns=['sepal_length','sepal_width','petal_length','petal_width'])
```

```
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300,random_state=0)
kmeans.fit(irisDF)
```

pca로 2개 차원으로 축소하여 시각화



- n_clusters: 군집화할 개수(군집 중심점의 개수)
- init: 초기에 군집 중심점의 좌표 설정 방식. 일반적으로 k-means++ 방식 사용 (디폴트값= k-means++)
- max_iter: 최대 반복 횟수, 이 횟수 이전에 모든 데이터의 중심점 이동이 없으면 종료 (디폴트값=300)

2.4 사이킷런을 이용한 k-평균 군집화

실루엣 계수

```
# irisDF에 cluster 결과 추가
irisDF['cluster'] = kmeans.labels_
```
각 데이터포인트가 속한 군집 중심점 레이블

```
# 실루엣 분석 metric 값을 구하기 위한 API 추가
from sklearn.metrics import silhouette_samples, silhouette_score

# iris 의 모든 개별 데이터에 실루엣 계수값을 구함.
score_samples = silhouette_samples(iris.data, irisDF['cluster'])
print('silhouette_samples( ) return 값의 shape' , score_samples.shape)  # (150,)

# irisDF에 실루엣 계수 컬럼 추가
irisDF['silhouette_coeff'] = score_samples

# 모든 데이터의 평균 실루엣 계수값을 구함.
average_score = silhouette_score(iris.data, irisDF['cluster'])
print('붓꽃 데이터셋 Silhouette Analysis Score:{0:.3f}'.format(average_score))  # 0.553
```

silhouette_samples(X, labels)
: 각 데이터 포인트의 실루엣 계수를 계산해 반환

silhouette_score(X, labels) (=np.mean(silhouette_samples())
: 전체 데이터의 실루엣 계수 값을 평균해 반환함.
이 값이 높을수록 군집화가 어느정도 잘됐다고 판단 가능.
하지만 무조건 이 값이 높다고 해서 군집화가 잘됐다고 판단할 수는 없음.

```
irisDF.groupby('cluster')['silhouette_coeff'].mean()
```

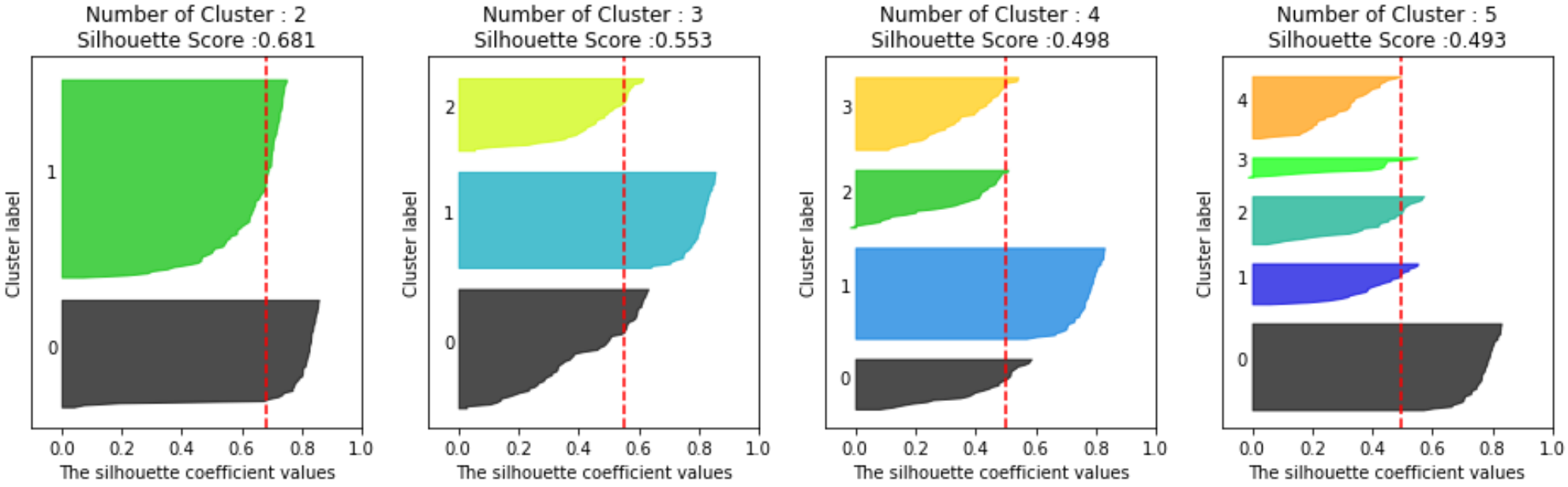
```
cluster
0    0.417182
1    0.797630
2    0.451105
```

➔ 군집별 평균 실루엣 계수 값
0번과 2번은 1번에 비해 평균값이 상대적으로 낮음.

2.4 사이킷런을 이용한 k-평균 군집화

실루엣 계수 시각화 → 군집 개수 최적화

전체 데이터의 평균 실루엣 계수 값이 높다고 해서 반드시 최적의 군집 개수로 군집화가 잘 됐다고 볼 수 없음
(특정 군집 내의 실루엣 계수값만 너무 높고, 다른 군집의 실루엣 값이 낮아져도 평균적으로 높은 값을 가질 수 있기 때문)
➔ 개별 군집별로 적당히 분리된 거리를 유지하면서도,
군집 내의 데이터가 서로 뭉쳐 있는 경우에 적절한 군집 개수가 설정됐다고 판단할 수 있음.

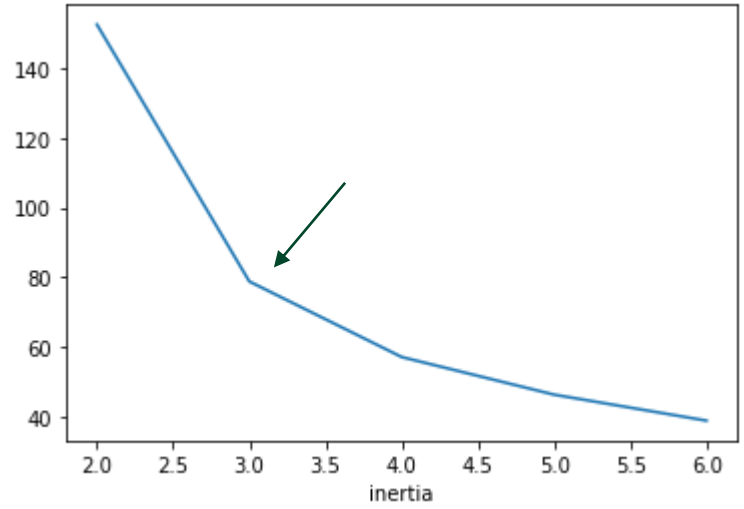


붓꽃 데이터 실루엣 계수 시각화
-> 군집 개수 2개로 하는 것이 좋아보임. 나머지는 군집 별 실루엣 계수의 편차가 큼.

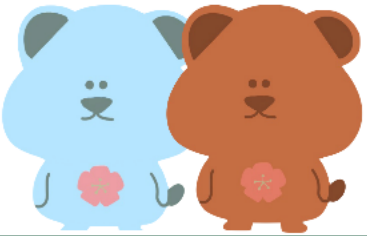
2.4 사이킷런을 이용한 k-평균 군집화

이너셔와 엘보우

```
inertia = []
for k in range(2, 7):
    km = KMeans(n_clusters = k, random_state = 0)
    km.fit(iris.data)
    inertia.append(km.inertia_)
plt.plot(range(2, 7), inertia)
plt.xlabel('k')
plt.xlabel('inertia')
plt.show()
```



03. 평균 이동



EWHA EURON

3.1 평균 이동 군집화

평균 이동 군집화

k-평균
중심에 소속된 데이터의 평균 거리 중심으로 이동

데이터가 모여 있는 밀도가 가장 높은 곳으로 군집의 중심점을 이동하면서 군집화 수행하는 기법
→ 데이터의 분포도를 이용해 군집 중심점을 찾음.
➔ 가장 집중적으로 데이터가 모여있어 확률 밀도 함수가 피크인 점을 군집 중심점으로 선정하며,
주어진 모델의 확률 밀도 함수를 찾기 위해 KDE를 이용함.

KDE: 커널 함수를 통해 어떤 변수의 확률 밀도 함수를 추정하는 대표적인 방법
개별 관측 데이터에 커널 함수를 적용한 값을 모두 더한 뒤, 데이터 건수로 나눠 확률 밀도 함수를 추정하는 커널 함수
→ 가우시안 분포 함수 사용함.
커널이란 분포를 찾을 때 사용되는 기본 단위(e.g. 히스토그램은 사각형 모양의 커널 사용하는 추정 방법)
히스토그램은 가장 간단한 비모수 추정방법이지만, 추정된 모양이 불연속이라는 단점이 있어,
이를 부드러운 형태의 밀도함수 추정하기 위해 사각형 kernel 대신 정규분포 사용하는 방법 → Gaussian Kernel

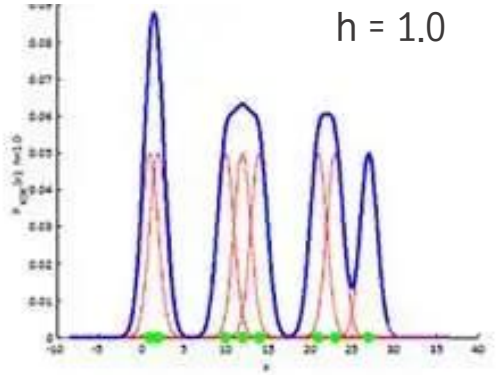
$$KDE = \frac{1}{n}\sum_{i}^{n} K_h\left(x - x_i\right) = \frac{1}{nh}\sum_{i}^{n} K\left(\frac{x - x_i}{h}\right)$$

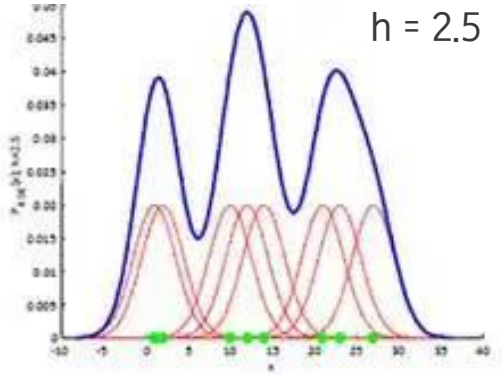
K :커널 함수
x : 확률 변수값
xi : 관측값
h : 대역폭
대역폭 h는 KDE 형태를 부드러운 형태로 평활화하는데 적용됨

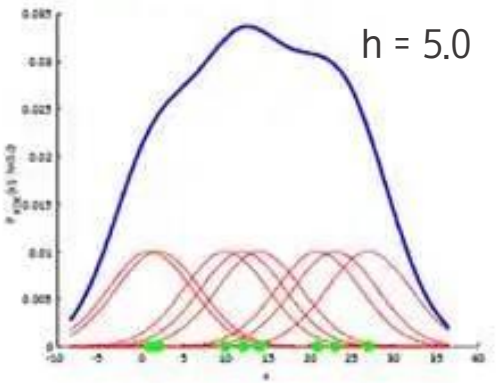
3.1 평균 이동

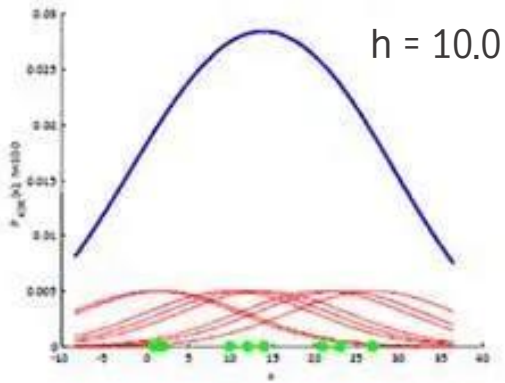
KDE

h 설정 → 확률 밀도 추정 성능 크게 좌우함









h↓ → 좁고 뾰족한 KDE → 변동성이 큰 방식으로 확률 밀도 추정
→ 군집 중심점 수↑, 과대적합 가능성

h↑ → 과도하게 평활화됨 → 단순화된 방식으로 확률밀도 추정
→ 군집 중심점↓, 과소적합 가능성

➔ 적절한 KDE의 대역폭 h를 계산하는 것 평균 이동 군집화에 매우 중요
(= 사이킷런의 bandwidth를 최적화 값으로 설정하는 것 매우 중요)

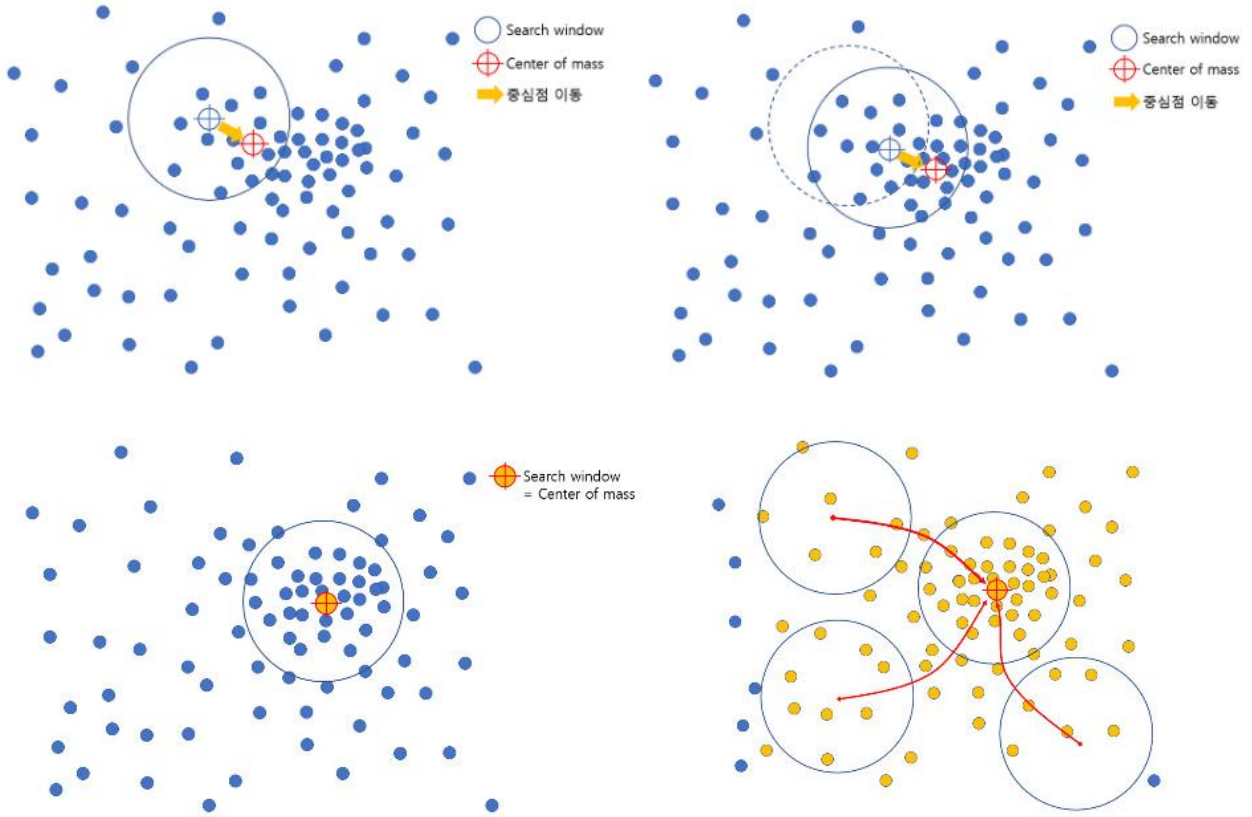
```
from sklearn.cluster import MeanShift
meanshift= MeanShift(bandwidth=1) 파라미터 bandwidth = 대역폭 h
cluster_labels = meanshift.fit_predict(X)
print('cluster labels 유형:', np.unique(cluster_labels))
```

3.1 평균 이동

평균 이동 원리(과정)

특정 데이터를 반경 내의 데이터 분포 확률 밀도가 가장 높은 곳으로 이동하기 위해,
주변 데이터와의 거리 값을 KDE 함수 값으로 입력한 뒤, 그 반환 값을 현재 위치에서 업데이트하면서 이동하는 방식을 취함.
이러한 방식을 전체 데이터에 반복적으로 적용하면서 데이터의 군집 중심점을 찾아냄

1. 개별 데이터의 특정 반경 내에
   주변 데이터를 포함한 데이터 분포를
   KDE 기반의 Mean Shift 알고리즘으로 계산
2. KDE로 계산된 데이터 분포도가 높은 방향으로
   데이터 이동
3. 모든 데이터를 1~2까지 수행하면서 데이터 이동.
   개별 데이터들이 군집중심점으로 모임.
4. 지정된 반복 횟수만큼
   전체 데이터에 대해 군집화 수행
5. 개별 데이터들이 모인 중심점을
   군집 중심점으로 설정



3.2 사이킷런을 이용한 평균 이동 군집화

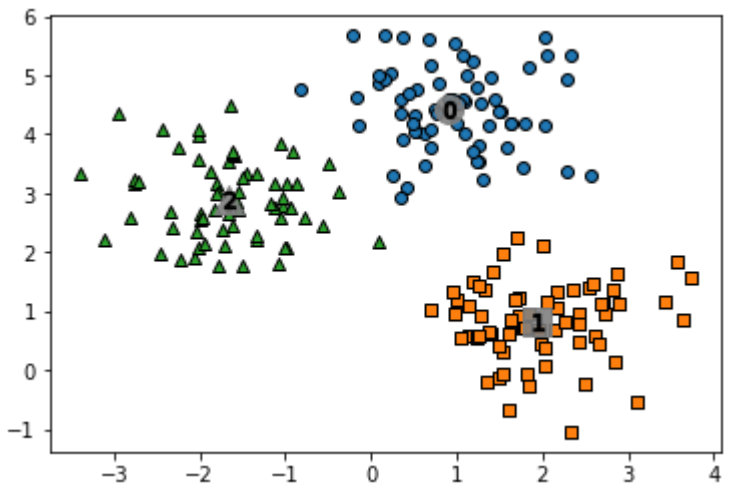
```
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.cluster import MeanShift

# 데이터셋 생성
X, y = make_blobs(n_samples=200, n_features=2, centers=3,
                  cluster_std=0.7, random_state=0)

meanshift= MeanShift(bandwidth=0.8)
# 결과값 변수에 저장
cluster_labels = meanshift.fit_predict(X)
# 군집화 된 개수를 넘파이의 unique를 통해서 파악
print('cluster labels 유형:', np.unique(cluster_labels))
```

파라미터 bandwidth = 대역폭 h

```
from sklearn.cluster import estimate_bandwidth

bandwidth = estimate_bandwidth(X)
print('bandwidth 값:', round(bandwidth,3))  # 1.816
```

estimate_bandwidth(피처 데이터 세트)
: 최적화된 bandwidth 값 반환

최적화된 bandwidth를
평균 이동 입력값으로 적용해 군집화 수행한 후 시각화




```
print(clusterDF.groupby('target')['meanshift_label'].value_counts())
```

```
target  meanshift_label
0       0                 67
1       1                 67
2       2                 66
Name: meanshift_label, dtype: int64
```

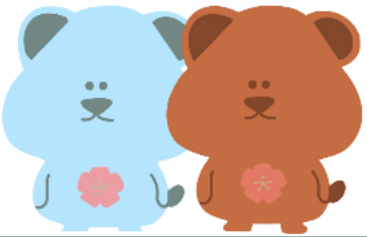
→타겟값과 군집 label 값이 1:1로 잘 매칭됨

3.3 평균 이동 알고리즘 장단점

평균 이동 알고리즘 장단점

장점	단점
데이터 세트의 형태를 특정 형태로 가정하거나, 특정 분포도 기반의 모델로 가정하지 않음 → 유연한 군집화 가능	알고리즘의 수행 시간이 오래 걸림
이상치의 영향력 크지 않음	bandwidth 크기에 따른 군집화 영향도가 매우 큼 → 컴퓨터 비전 영역에서 많이 사용됨
미리 군집의 개수 정할 필요 없음	

04. 병합군집

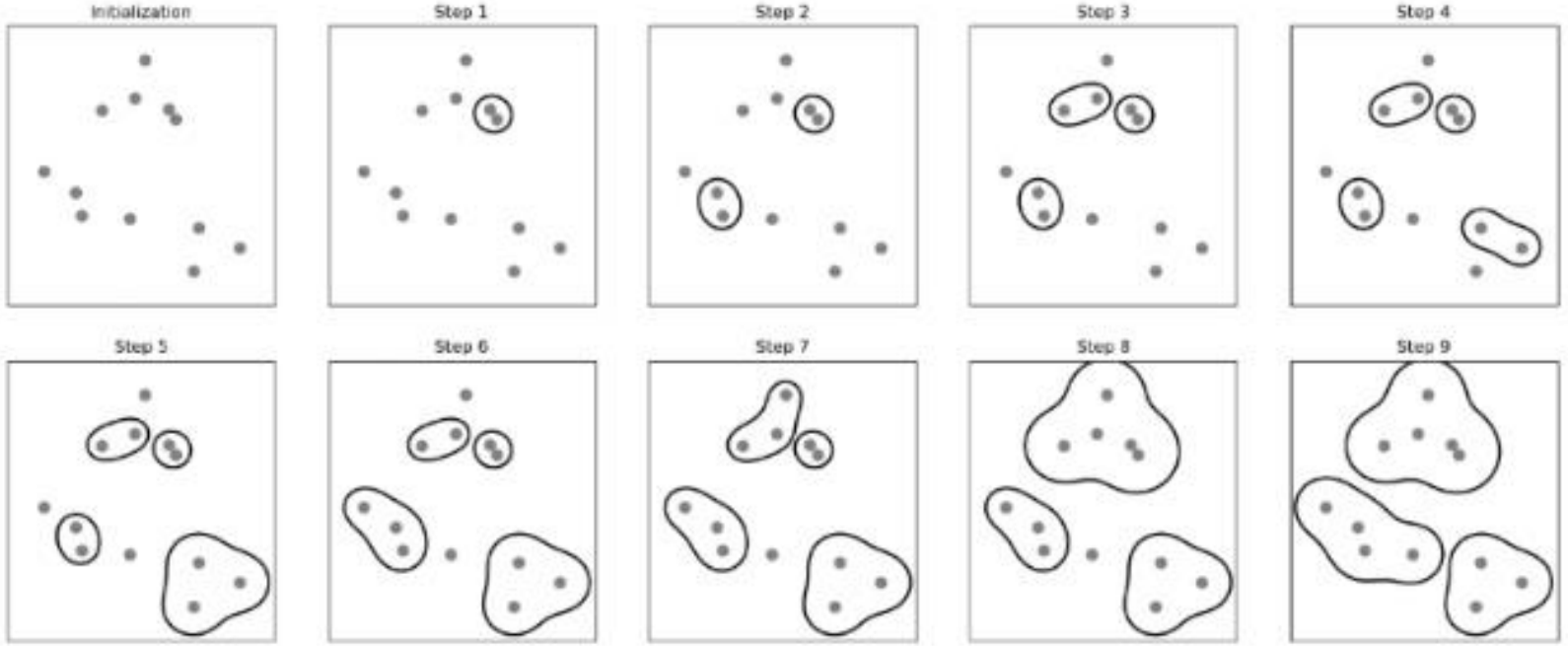


EWHA EURON

4.1 병합 군집

병합 군집(Agglomerative Clustering) :

각각의 데이터 포인트를 하나의 클러스터로 지정하고 지정된 개수의 클러스터가 남을 때까지 가장 비슷한
두 클러스터를 합쳐 나가는 알고리즘

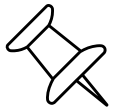


1. 초기에 각 포인터가 하나의 클러스터
2. 가장 가까운 두 클러스터가 합쳐짐
3. Step4 까지 이 방식으로 진행
4. Step5에서 두 개의 포인트를 가진
   클러스터 중 하나가 3개로 확장
5. Step9까지 이 방식으로 진행
6. 지정된 클러스터 개수가 되면
7. 알고리즘은 종료

* 이러한 병합 군집 알고리즘 작동
특성상 새로운 데이터 포인트에 대해서는
예측을 할 수 없음

EWHA EURON

4.1 병합 군집



두 클러스터를 합쳐 나가는 방식

-Ward: 모든 클러스터 내의 분산을 가장 작게 증가시키는 두 클러스터를 합치는 방식, 크기가 비교적 비슷한 클러스터가 만들어짐
-Average: 클러스터 포인트 사이의 평균 거리가 가장 짧은 두 클러스터를 합치는 방식
-Complete: 클러스터 포인트 사이의 최대 거리가 가장 짧은 두 클러스터를 합치는 방식

* 클러스터에 속한 포인트 수가 많이 다를 때(하나의 클러스터가 다른 것 보다 매우 클 때)
average 나 complete가 더 나음

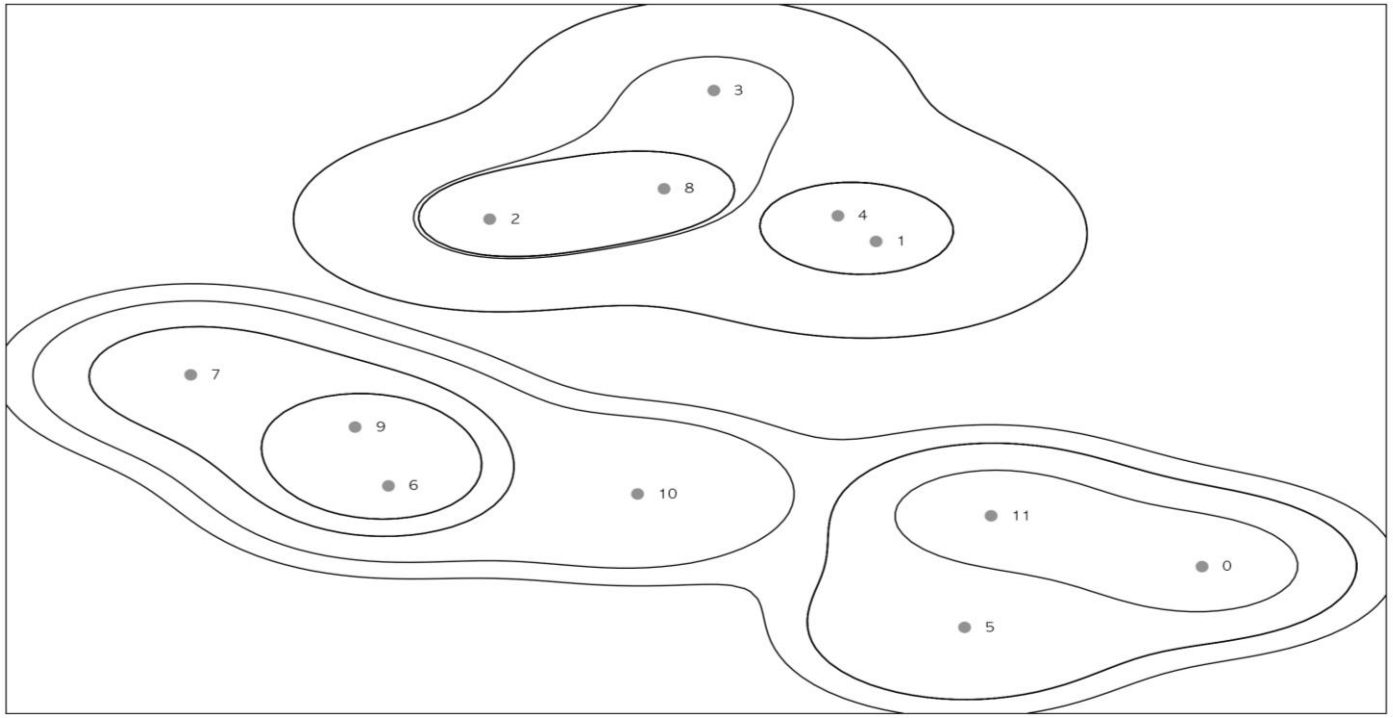
-> 병합 군집은 계층적 군집(Hierarchical Clustering) 을 만듬
군집이 반복하여 진행되면 모든 포인트는 하나의 포인트를 가진 클러스터에서 시작하여
마지막 클러스터까지 이동하게 됨
즉, 작은 클러스터들이 모여 큰 클러스터를 이루는 계층적 구조를 가지는 것

4.2 계층적 군집

계층적 군집(Hierarchical Clustering):

계층적 트리 모형을 이용하여 개별 데이터 포인트들을 순차적, 계층적으로 유사한 클러스터로 통합하여 군집화를 수행하는 알고리즘
K means 알고리즘과는 달리 클러스터의 개수를 사전에 정하지 않아도 학습 수행 가능

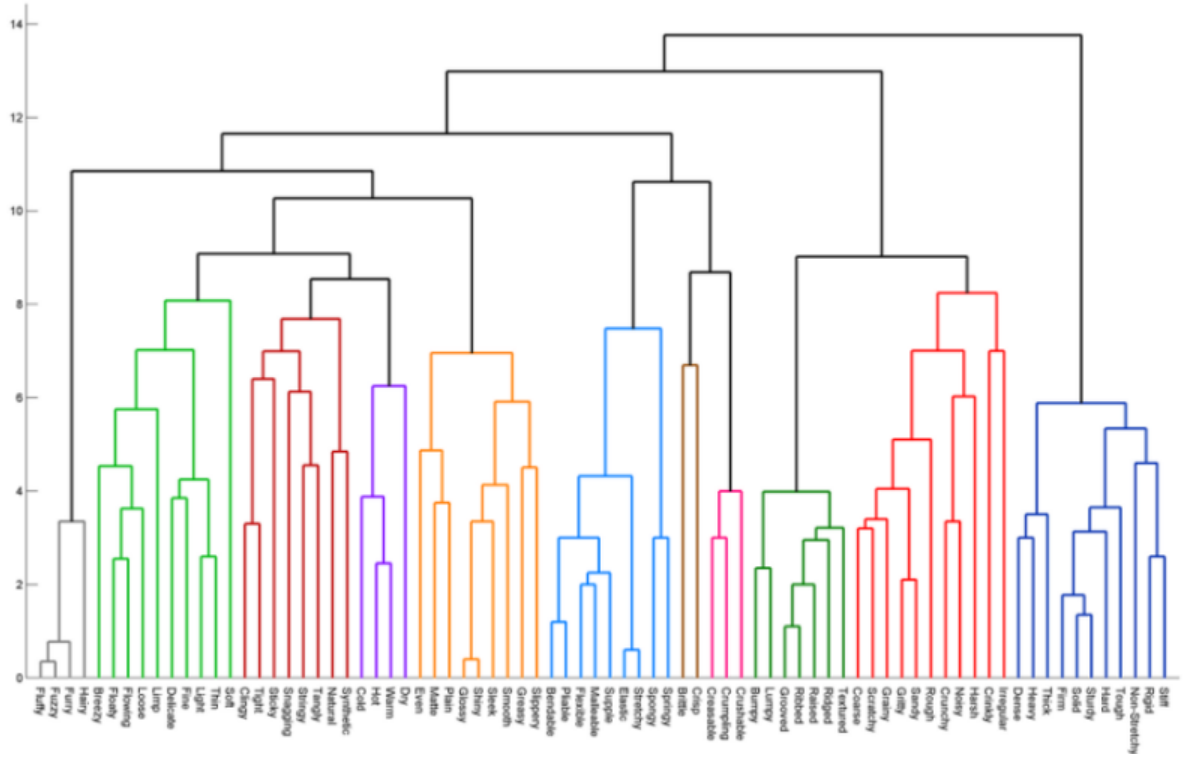
- 상향식 병합 군집 방식(Agglomerative Clustering): 개개의 포인트를 순차적으로 병합
- 하향식 병합 군집 방식(Divisive clustering): 전체를 하나의 클러스터로 하여 그것을 분할해나감



4.3 덴드로그램

덴드로그램으로 3차원 이상의 데이터 시각화

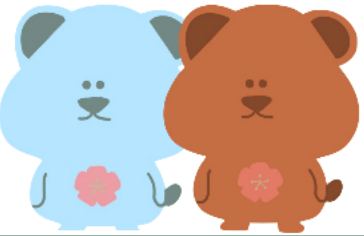
덴드로그램(dendrogram): 계층 군집을 시각화하는 도구로, 다차원 데이터 세트를 처리 할 수 있음
계층적 군집에서 클러스터의 개수를 지정해주지 않아도 학습을 수행할 수 있는 것은 개체들이 결합되는 순서를 나타내는 트리 형태의 구조인 덴드로그램 덕분
덴드로그램에서 가지의 길이는 합쳐진 클러스터가 얼마나 멀리 떨어져 있는지 보여줌



덴드로그램을 가로 선으로 분할하면 클러스터를 임의로 나눌 수 있음
즉, 군집화를 하기 전 클러스터 개수를 정해줘야하는 k means 알고리즘과 달리 병합 군집같은 계층적 군집 알고리즘은 군집화를 완료한 후에 사용자가 시각화된 덴드로 그램을 보고 클러스터를 나눌 수 있는 것
그러나 k means 와 마찬가지로 데이터 간의 거리를 기반으로 하기 때문에 복잡한 형상의 데이터 세트는 구분하지 못함

EWHA EURON

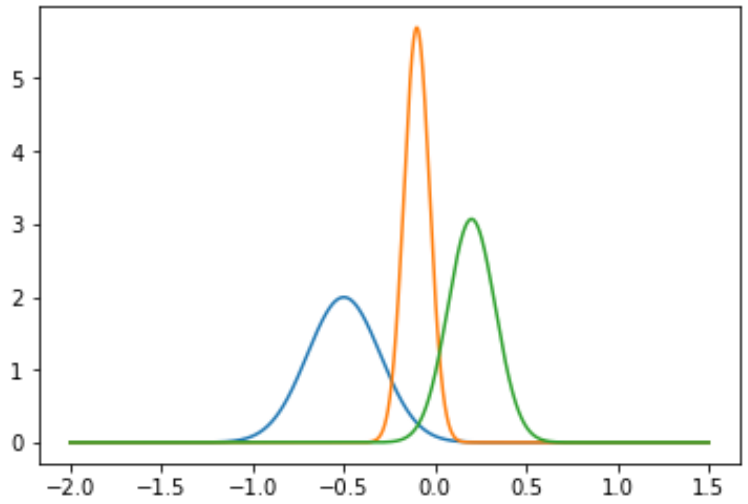
05. GMM



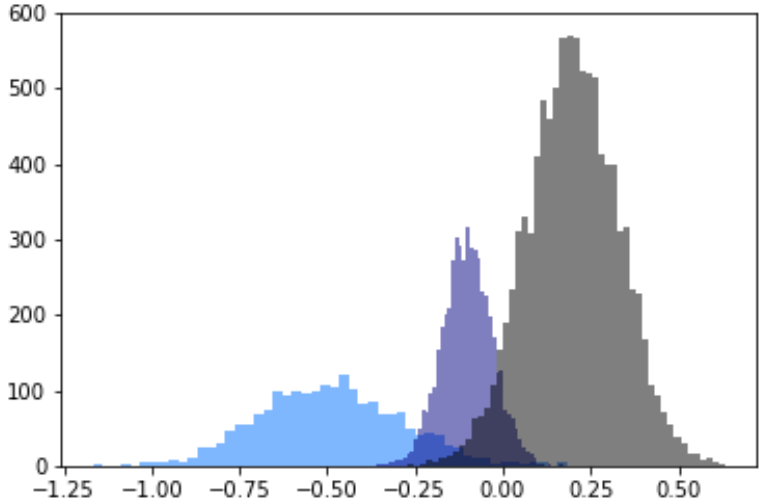
EWHA EURON

5.1 GMM

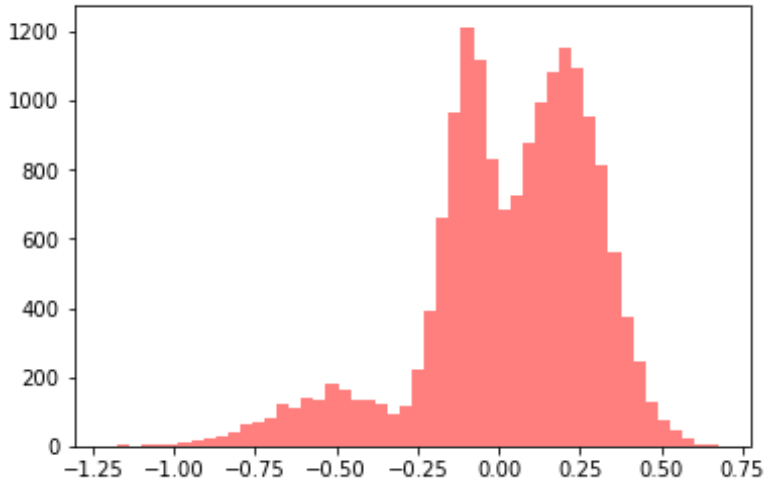
GMM(Gaussian Mixture Model) :
군집화를 적용하고자 하는 데이터가 여러 개의 가우시안 분포를 가진 데이터 집합이 섞여서 생성된 것으로 간주하고 군집화를 수행(확률 기반 군집화)



<3개의 정규분포의 pdf>



<3개의 정규분포로부터 랜덤하게 생성된 데이터>



<실제로 보게 되는 데이터>

정규분포 1: 평균 -0.5 표준편차 0.2
정규분포 2: 평균 -0.1 표준편차 0.07
정규분포 3: 평균 0.2 표준편차 0.13

모수추정: 개별 정규 분포 찾기 (평균과 분산) & 데이터가 특정 정규 분포에 해당될 확률 구하기

5.2 GMM 파라미터

n_components : gaussian mixture 모델의 총 개수
(k 평균의 n_clusters와 같이 군집 개수를 정하는 데 중요한 역할 수행)

```
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=3, random_state=0).fit(iris.data)
gmm_cluster_labels = gmm.predict(iris.data)

# 클러스터링 결과를 irisDF 의 'gmm_cluster' 컬럼명으로 저장
irisDF['gmm_cluster'] = gmm_cluster_labels
irisDF['target'] = iris.target

# target 값에 따라서 gmm_cluster 값이 어떻게 매핑되었는지 확인.
iris_result = irisDF.groupby(['target'])['gmm_cluster'].value_counts()
print(iris_result)
```

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

```
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300,random_state=0).fit(iris.data)
kmeans_cluster_labels = kmeans.predict(iris.data)
irisDF['kmeans_cluster'] = kmeans_cluster_labels
iris_result = irisDF.groupby(['target'])['kmeans_cluster'].value_counts()
print(iris_result)
```

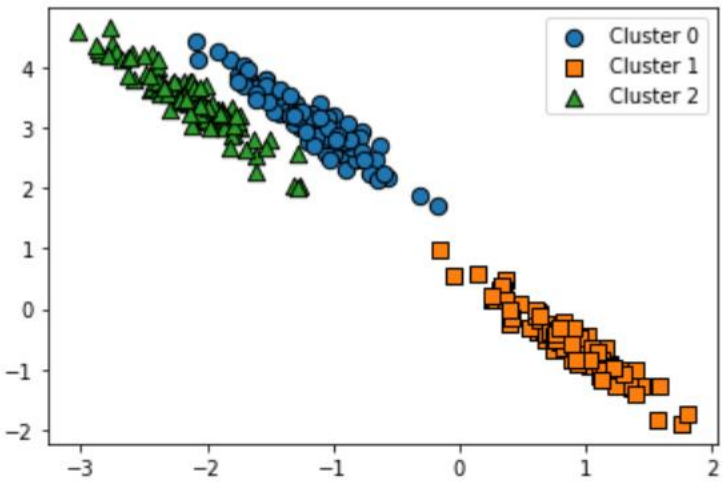
```
target  kmeans_cluster
0       1                50
1       0                48
        2                 2
2       2                36
        0                14
Name: kmeans_cluster, dtype: int64
```

EWHA EURON

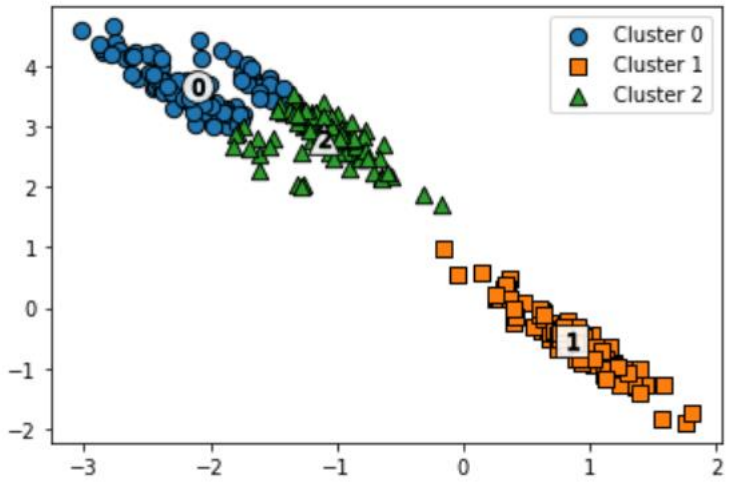
5.3 GMM vs K-Means

GMM vs K-평균

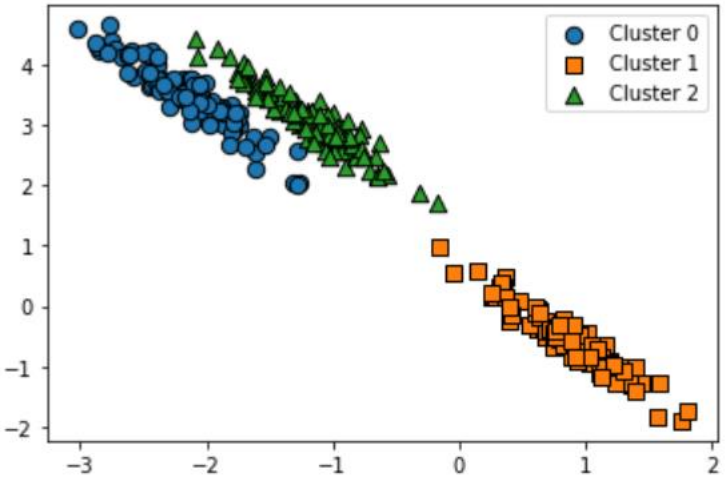
Kmeans는 원형의 범위에서 군집화를 수행, 대표적으로 데이터가 길쭉한 타원형으로 늘어선 경우에는 군집화를 잘 수행하지 못함



Make_blobs( )로 300개의 데이터 세트, 3개의 군집 세트, Cluster_std=0.5 를 만들고 길게 늘어난 타원형의 데이터 세트를 생성하기 위해 변환한후 target 별로 다른 마커로 표시해 시각화



<Kmeans로 군집화 수행 결과>
주로 원형 영역 위치로 개별 군집화되며 원하는 방향으로 구성되지 않음
Kmeans가 같은 거리상 원형으로 군집을 구성하며 위처럼 길쭉한 방향으로 데이터가 밀접한 경우 최적의 군집화 어려움



<GMM으로 군집화 수행 결과>
데이터가 분포된 방향에 따라 정확하게 군집화됨
K 평균과 다르게 군집의 중심 좌표를 구할 수 없기 때문에 군집 중심 표현이 시각화되지 않음

5.3 GMM vs K-Means

```
### KMeans Clustering ###
target  kmeans_label
0       2                73
        0                27
1       1               100
2       0                86
        2                14
Name: kmeans_label, dtype: int64

### Gaussian Mixture Clustering ###
target  gmm_label
0       2             100
1       1             100
2       0             100
Name: gmm_label, dtype: int64
```

< GMM과 Kmeans 군집화 효율 차이 비교 >

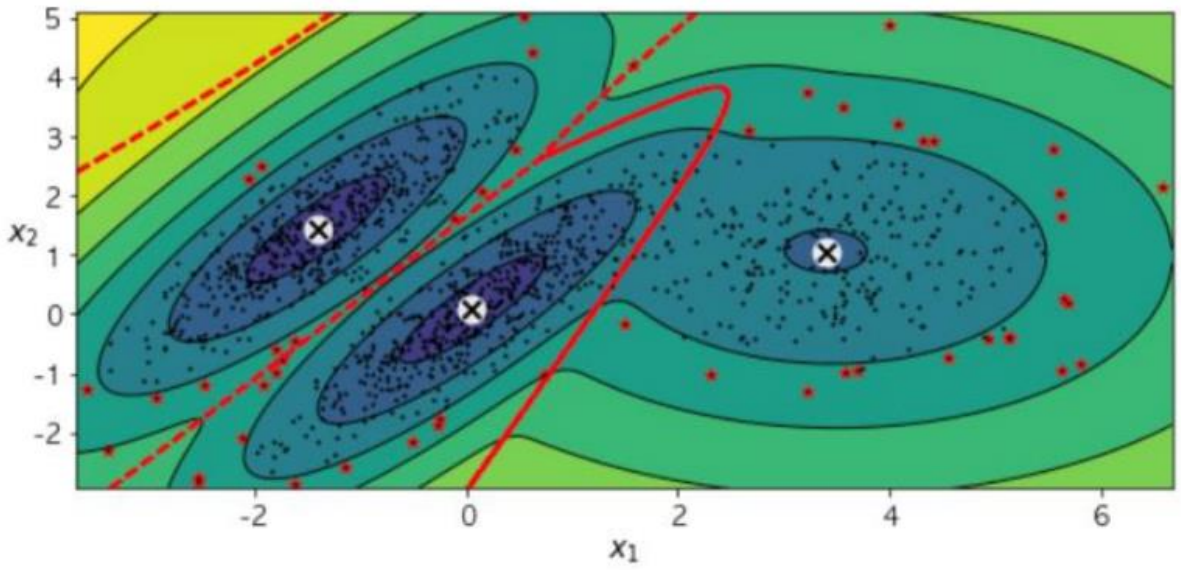
GMM은 Kmeans 보다 유연하게 다양한 데이터 세트에 잘 적용될 수 있다는 장점
군집화를 위한 수행 시간이 오래걸린다는 단점

5.4 이상치 탐지

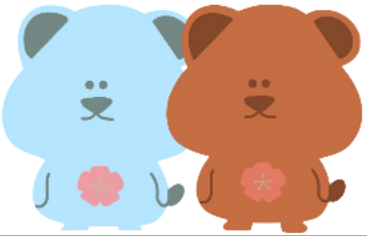
이상치 탐지 (outlier detection) :

보통과 많이 다른 샘플을 감지하는 작업 (이상치 <-> 정상치)
훈련 데이터 셋에 이상치가 포함됨
밀도 임계 값을 정해 밀도가 낮은 지역에 있는 모든 샘플을 이상치로 판단

```
X1, y1 = make_blobs(n_samples=1000, centers=((4, -4), (0, 0)), random_state=42)
X1 = X1.dot(np.array([[0.374, 0.95], [0.732, 0.598]]))
X2, y2 = make_blobs(n_samples=250, centers=1, random_state=42)
X2 = X2 + [6, -8]
X = np.r_[X1, X2]
y = np.r_[y1, y2]
gm = GaussianMixture(n_components=3, n_init=10, random_state=42)
gm.fit(X)
densities = gm.score_samples(X)
density_threshold = np.percentile(densities, 4)
anomalies = X[densities < density_threshold]
plt.figure(figsize=(8, 4))
plot_gaussian_mixture(gm, X)
plt.scatter(anomalies[:, 0], anomalies[:, 1], color='r', marker='*')
plt.ylim(top=5.1)
save_fig("mixture_anomaly_detection_plot")
plt.show()
```



06. DBSCAN

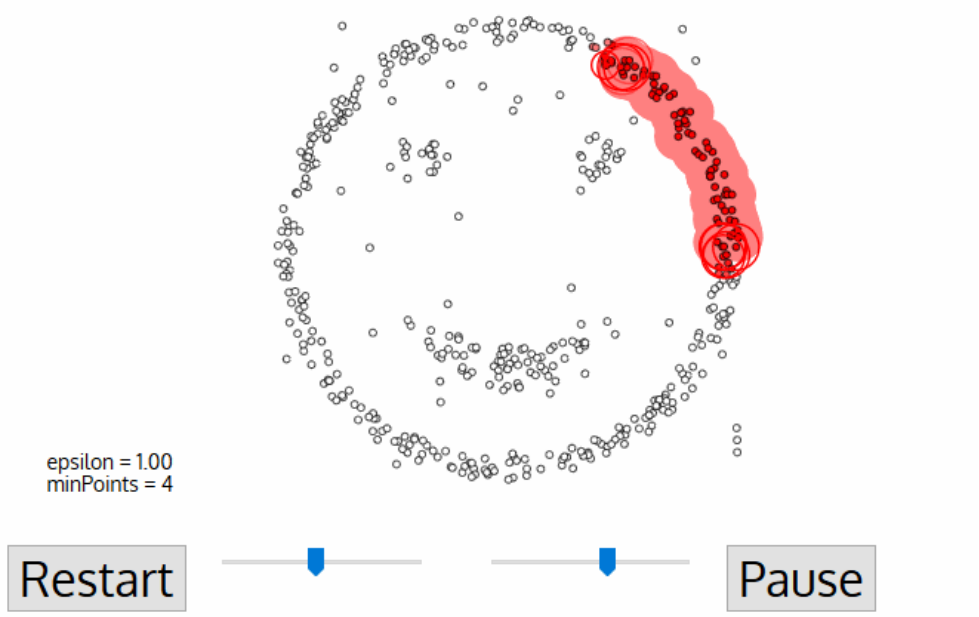


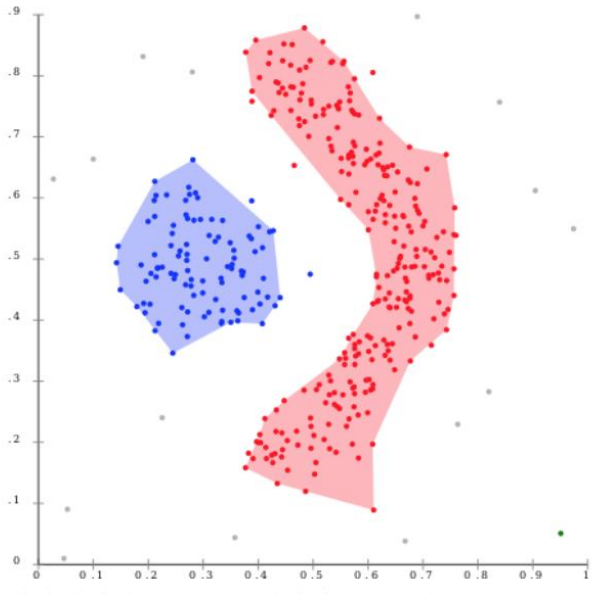
EWHA EURON

6.1 DBSCAN

DBSCAN(Density Based Spatial Clustering of Applications with Noise) :

데이터의 분포가 기하학적으로 복잡한 데이터 세트에도 효과적인 군집화가 가능(밀도 기반 군집화)
점이 세밀하게 몰려 있어서 밀도가 높은 부분을 클러스터링 하는 방식





DBSCAN은 core point 끼리 묶이기 때문에 k-means와는 전혀 다른 클러스터링
을 만들 수 있는데 위와 같은 모양의 클러스터가 가능해짐

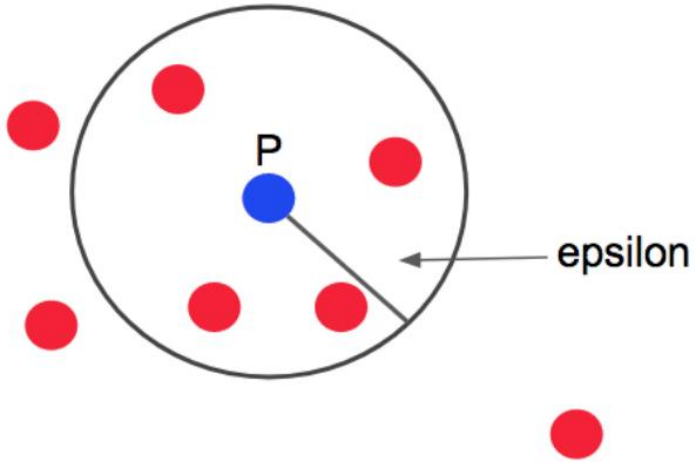
6.2 DBSCAN 파라미터

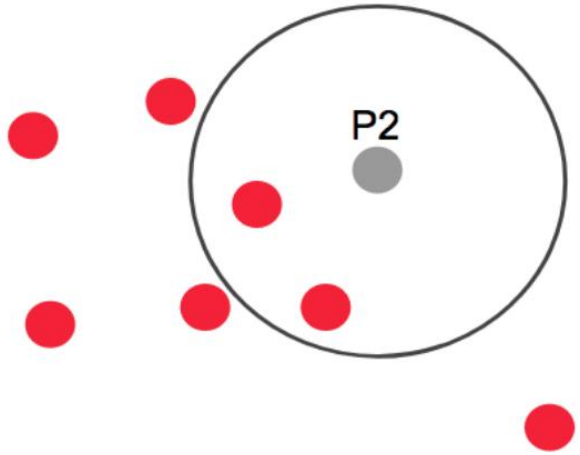
가장 중요한 두 가지 파라미터

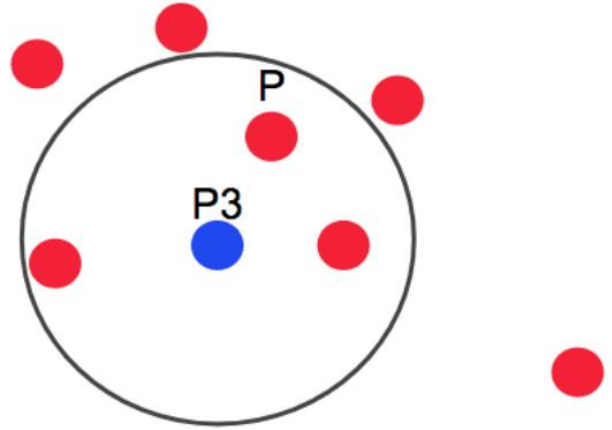
-주변 영역(epsilon): 개별 데이터를 중심으로 입실론 반경을 가지는 원형 영역
-최소 데이터 개수(min points): 개별 데이터의 입실론 주변 영역에 포함되는 타 데이터 개수

· 핵심 포인트(core point): 주변 영역 내에 최소 데이터 개수 이상의 타 데이터 가지고 있을 경우
· 이웃 포인트(neighbor point): 주변 영역 내에 위치한 타 데이터
· 경계 포인트(border point): 주변 영역 내에 최소 데이터 개수 이상의 이웃포인트를 가지고 있지 않지만 핵심 포인트를 이웃포인트로 가지고 있는 데이터
· 잡음 포인트(noise point): 최소 데이터 개수 이상의 이웃포인트를 가지고 있지 않으며, 핵심 포인트도 이웃 포인트로 가지고 있지 않는 데이터

6.3 DBSCAN 원리





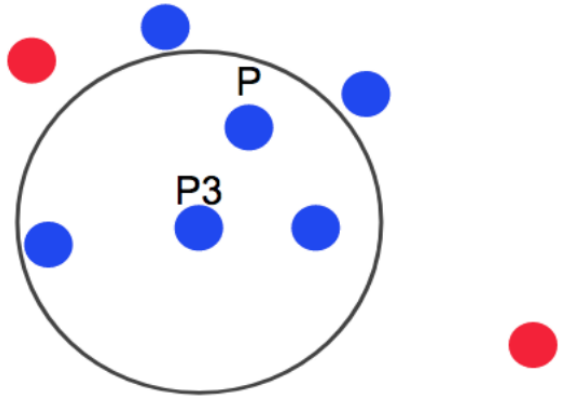


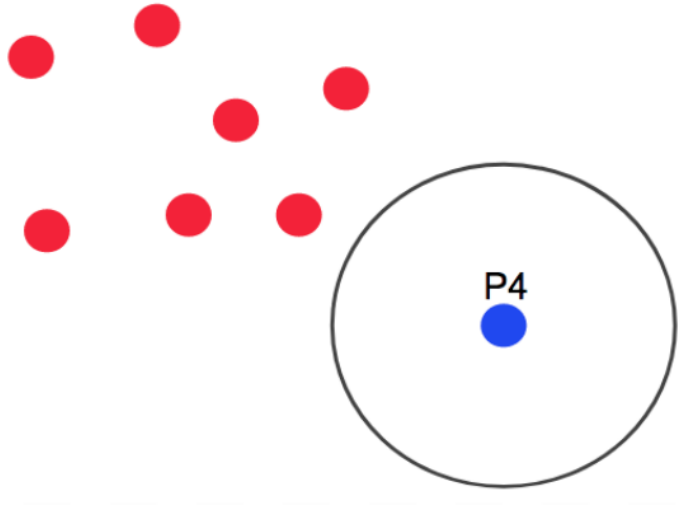
최소 데이터 개수를 4개로 하면 P를 기준으로 입실론 반경 내에 점이 4개 이상 있으면 하나의 군집으로 판단 -> 점이 5개 있기 때문에 하나의 군집으로 판단되고 P는 핵심포인트(core point)가 됨

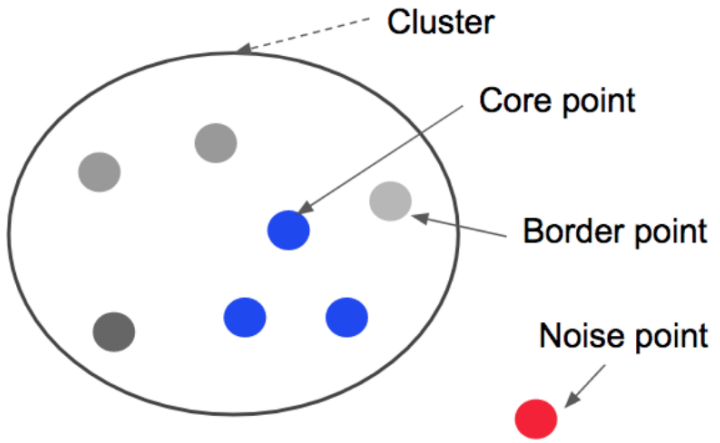
P2의 경우 입실론 반경 내 점이 3개 이므로 군집의 중심이 되는 핵심 포인트는 되지 못하지만 핵심포인트인 P를 이웃 포인트로 가지고 있기 때문에 P2는 경계 포인트가 됨

P3는 입실론 반경 내 점 4개를 가지고 있기 때문에 핵심 포인트가 됨

6.3 DBSCAN 원리







P3를 중심으로 하는 반경 내에 다른 핵심 포인트 P가 포함이 되어 있는데, 이 경우 P와 P3는 직접 접근이 가능
-> 직접 접근이 가능한 핵심 포인트를 서로 연결하면서 군집 영역을 확장해 나가는 것이 DBSCAN 군집화 방식

P4는 반경 내 최소 데이터를 가지고 있지도 않고 핵심 포인트 또한 이웃 데이터로 가지고 있지 않아 잡음 포인트가 됨

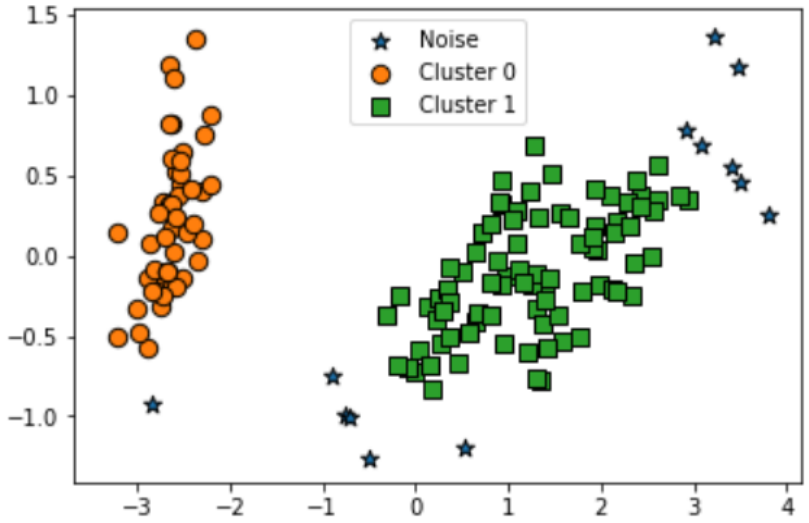
6.4 DBSCAN 적용하기

DBSCAN 알고리즘으로 붓꽃 데이터 세트 군집화

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.6, min_samples=8, metric='euclidean')
dbscan_labels = dbscan.fit_predict(iris.data)

irisDF['dbscan_cluster'] = dbscan_labels
irisDF['target'] = iris.target

iris_result = irisDF.groupby(['target'])['dbscan_cluster'].value_counts()
print(iris_result)
```

```
target  dbscan_cluster
0        0                49
        -1                 1
1        1                46
        -1                 4
2        1                42
        -1                 8
Name: dbscan_cluster, dtype: int64
```

0과 1 외에 특이하게 -1이 군집 레이블로 있는데 군집 레이블이 -1인 것은 노이즈에 속하는 군집 의미
-> 위 데이터 세트는 0과 1 두 개의 군집으로 군집화



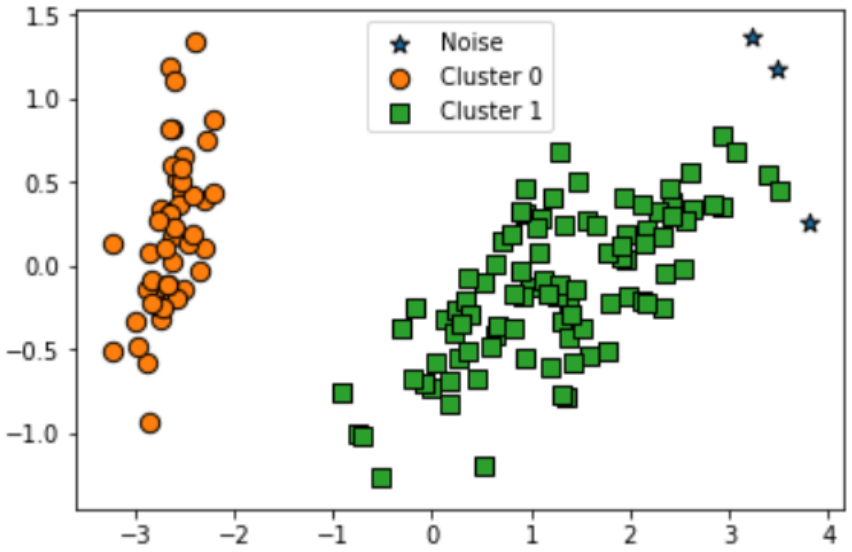
군집화 데이터 세트를 2차원 평면에서 표현하기 위해 PCA를 이용해 2개의 피처로 압축 변환한 뒤 시각화 한 것

6.4 DBSCAN 적용하기

```
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.8, min_samples=8, metric='euclidean')
dbscan_labels = dbscan.fit_predict(iris.data)

irisDF['dbscan_cluster'] = dbscan_labels
irisDF['target'] = iris.target

iris_result = irisDF.groupby(['target'])['dbscan_cluster'].value_counts()
print(iris_result)

visualize_cluster_plot(dbscan, irisDF, 'dbscan_cluster', iscenter=False)
```

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

일반적으로 eps의 값을 크게 하면 포함 하는 데이터가 많아지게 되므로 노이즈 데이터 개수는 작아짐
Min_samples을 크게 하면 주어진 반경 내에 더 많은 데이터를 포함해야하므로 노이즈 데이터 개수가 커지게 됨

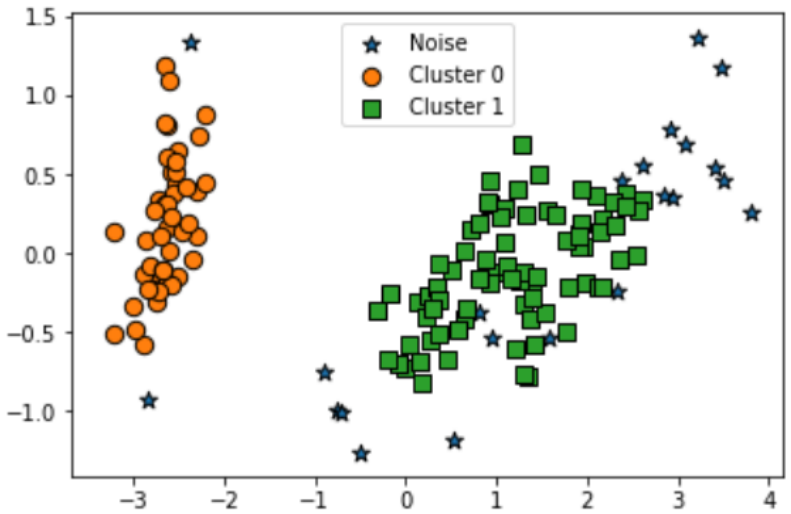


기존에 eps이 0.6일 때 노이즈로 분류된 데이터 세트는 eps 반경이 커지며 cluster1에 소속됨

6.4 DBSCAN 적용하기

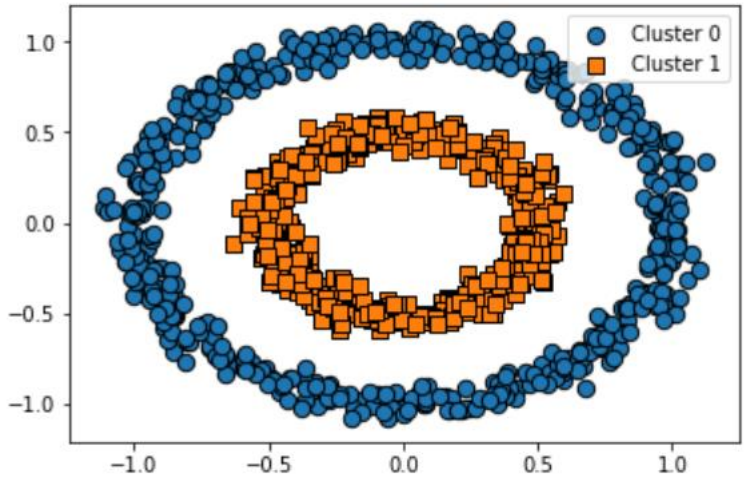
```
dbscan = DBSCAN(eps=0.6, min_samples=16, metric='euclidean')
dbscan_labels = dbscan.fit_predict(iris.data)

irisDF['dbscan_cluster'] = dbscan_labels
irisDF['target'] = iris.target

iris_result = irisDF.groupby(['target'])['dbscan_cluster'].value_counts()
print(iris_result)
visualize_cluster_plot(dbscan, irisDF, 'dbscan_cluster', iscenter=False)
```

```
target  dbscan_cluster
0        0               48
        -1                2
1        1               44
        -1                6
2        1               36
        -1               14
Name: dbscan_cluster, dtype: int64
```

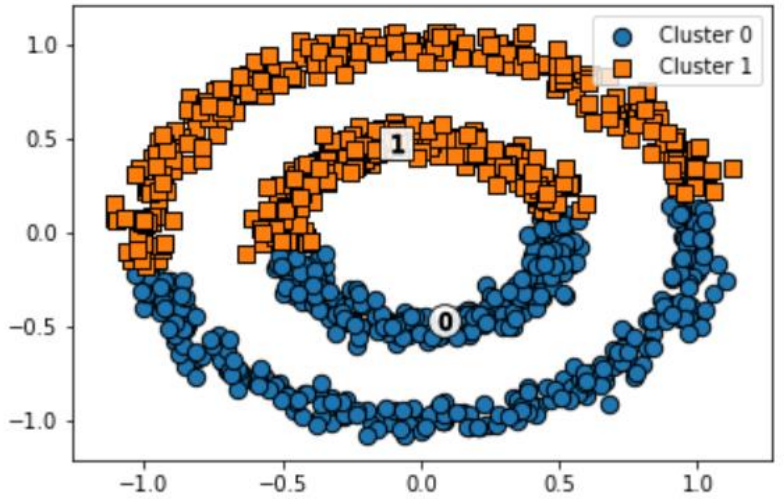


노이즈 데이터가 기존보다 많이 증가함

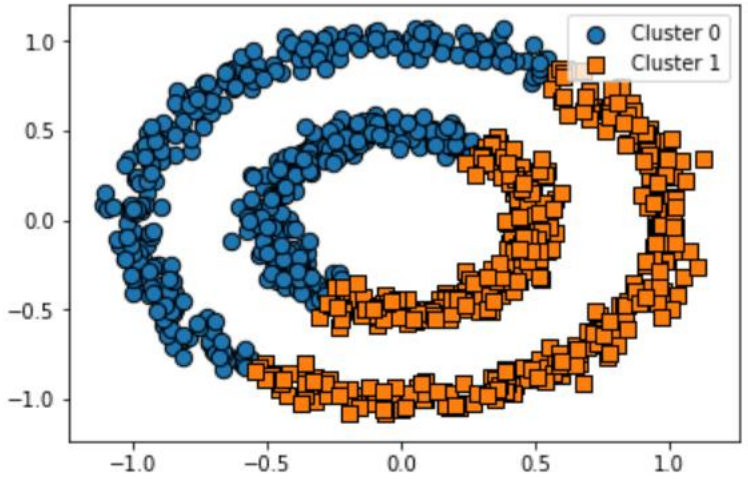
6.5 K-평균 vs GMM vs DBSCAN



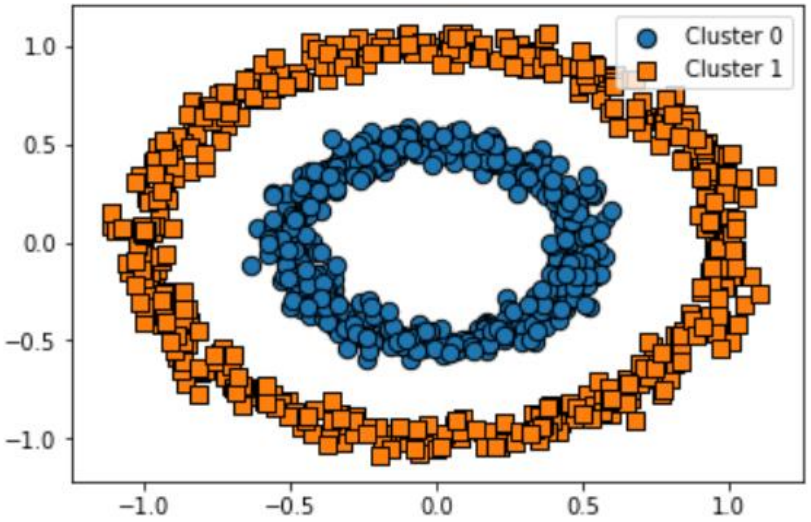
Make_circles( ) 함수로 내부 원과 외부 원 형태로 된 2차원 데이터 세트 생성



K-평균으로 군집화 한 결과
-> 위, 아래 군집 중심을 기반으로 위와 아래 절반으로 군집화됨

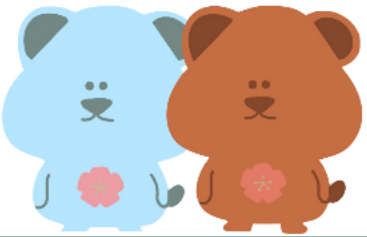


GMM 적용 결과
-> 원하는 방향으로 군집화 되지 않음



DBSCAN으로 군집화 적용
-> 원하는 방향으로 정확히 군집화 됨

07. 클러스터 개수 선택하기



EWHA EURON

7.1 AIC와 BIC

클러스터 개수 선택하기

- K-means에서 사용하는 이너셔, 실루엣 점수 등의 방식 = 클러스터가 타원형이거나 크기가 다르면 안정적 X

  → 가우시안 혼합 모델에서는 사용 X

- 그럼 어떤 방식을 사용할까?

AIC (Akaike Information Criterion)

- $\mathrm{AIC} = 2p - 2\log(\hat{L})$

BIC (Bayesian Information Criterion)

- $\mathrm{BIC} = \log(m)\,p - 2\log(\hat{L})$

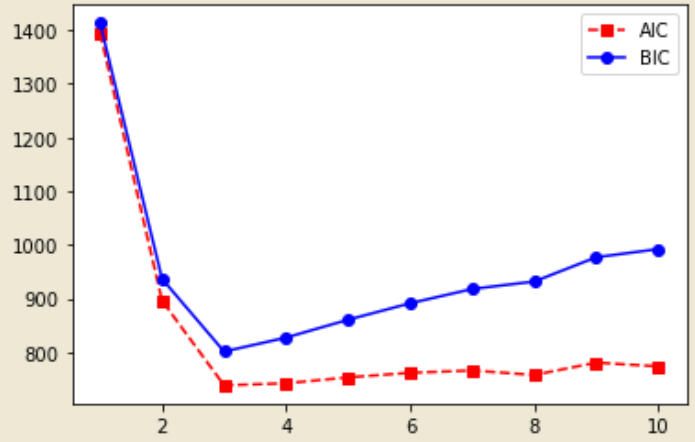
- m : 샘플의 개수
- p : 모델이 학습할 파라미터 개수
- $\hat{L}$: 모델의 가능도 함수의 최댓값

7.1 AIC와 BIC

```
from sklearn.mixture import GaussianMixture

gmm_per_k = [GaussianMixture(n_components=k, n_init=10, random_state=0).fit(X_aniso)
      for k in range(1, 11)]
```

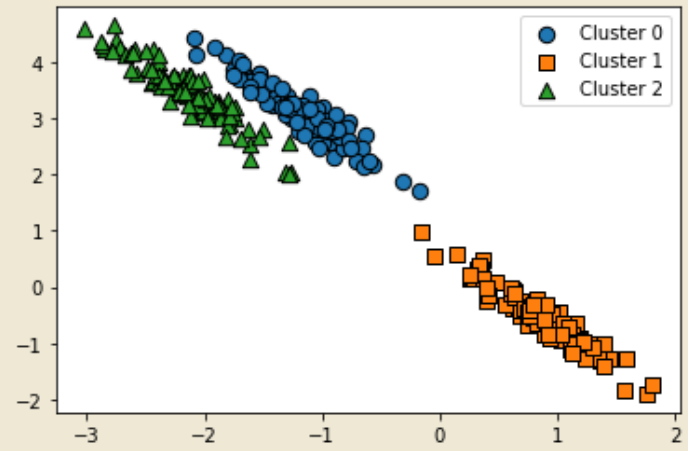
```
bics = [model.bic(X_aniso) for model in gmm_per_k]
aics = [model.aic(X_aniso) for model in gmm_per_k]
```

```
plt.plot(range(1,11), aics, 'rs--')
plt.plot(range(1,11), bics, 'bo-')
plt.legend(['AIC', 'BIC'])
plt.show()
```



→ AIC or BIC 값을 최소화하는 모델 선택
이 경우 k=3 !!

```
# 3개의 n_components기반 GMM을 X_aniso 데이터 셋에 적용
gmm = GaussianMixture(n_components=3, n_init=10, random_state=0)
gmm_label = gmm.fit(X_aniso).predict(X_aniso)
clusterDF['gmm_label'] = gmm_label

# GaussianMixture는 cluster_centers_ 속성이 없으므로 iscenter를 False로 설정.
visualize_cluster_plot(gmm, clusterDF, 'gmm_label',iscenter=False)
```



7.2 가능도 함수

가능도 함수 (likelihood function)

- 평소에 우리가 생각하는 확률 p(x | θ)

  = 파라미터 θ를 알고 있을 때, 그 모델이 x를 출력하는 것이 얼마나 그럴듯한가

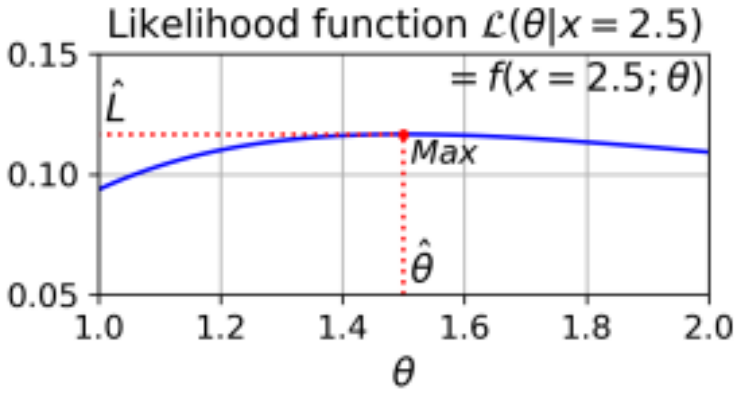
  = x의 함수

- 가능도 L(θ | x)

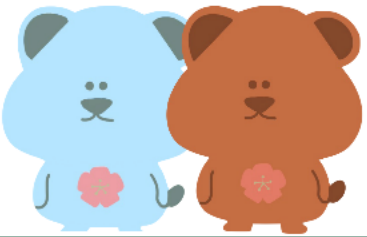
  = 관측된 데이터 x를 알고 있을 때, 특정 파라미터 θ가 얼마나 그럴듯한가

  = θ의 함수

- 가능도가 최대가 되는 파라미터 θ, 이 경우 클러스터 개수를 찾는 것



08. 베이즈가우시안 혼합 모델



EWHA EURON

8.1 베이즈가우시안 혼합 모델

베이즈가우시안 혼합 모델

- 불필요한 클러스터의 가중치를 0 또는 0에 가깝게 만드는 방식
- 최적의 클러스터 개수를 수동으로 찾을 필요 X

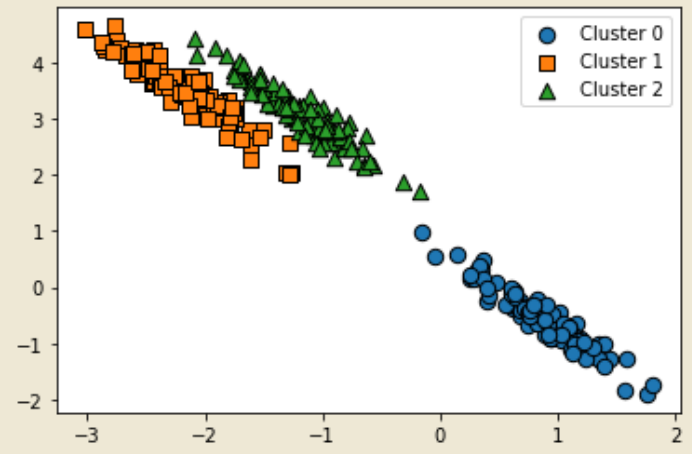
- BayesianGaussianMixture 클래스 사용
- n_components를 최적의 클러스터 개수보다 크게 지정

  → 자동으로 불필요한 클러스터 제거

```
from sklearn.mixture import BayesianGaussianMixture

bgm = BayesianGaussianMixture(n_components=10, n_init=10, random_state=0)
bgm.fit(X_aniso)
```

```
BayesianGaussianMixture(n_components=10, n_init=10, random_state=0)
```

```
bgm_label = bgm.fit(X_aniso).predict(X_aniso)
clusterDF['bgm_label'] = bgm_label

visualize_cluster_plot(bgm, clusterDF, 'bgm_label',iscenter=False)
```



8.1 베이즈가우시안 혼합 모델

```
np.round(bgm.weights_, 2)
```

```
array([0.34, 0.33, 0.33, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ])
```

가중치를 확인해보면, 지정한 10개의 클러스터 중 3개가 필요함을 감지

→ 나머지 클러스터의 가중치는 모두 0이 된 것을 확인할 수 있다.

잠재변수 z

- 클러스터 파라미터 (가중치, 평균, 공분산행렬 등)을 고정된 모델 파라미터가 아닌 잠재 확률 변수로 취급
- 클러스터 할당
- z는 클러스터 파라미터와 클러스터 할당을 모두 포함

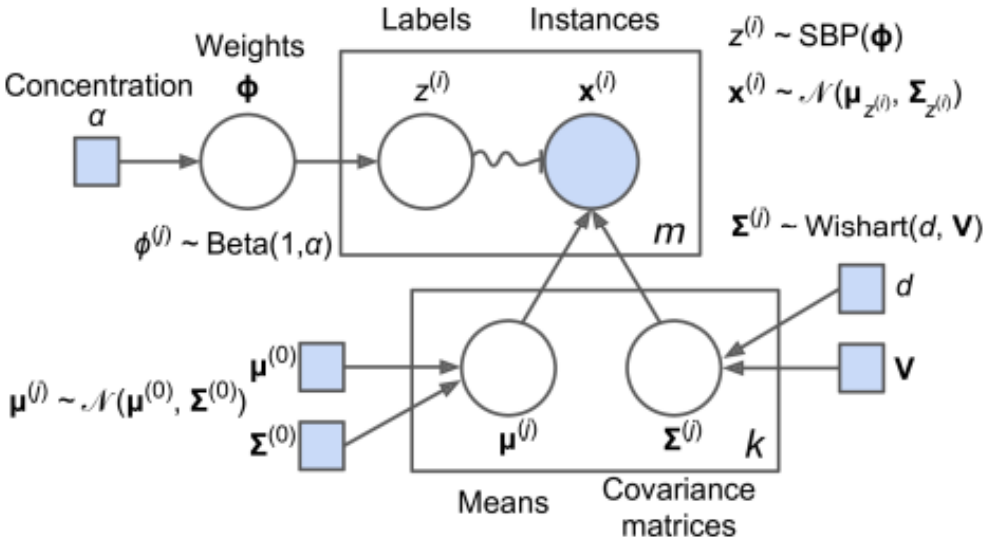
8.1 베이즈가우시안 혼합 모델

SBP (Stick-Breaking Process)

- φ = [0.3, 0.6, 0.5, …]를 가정하면 샘플의 30%가 클러스터0에 할당, 남은 샘플의 60%가 클러스터1에 할당, 남은 샘플의 50% 클러스터 2에 할당
- 농도 α가 크면 φ 값이 0에 가깝게 → 많은 클러스터 생성
- 농도 α가 작으면 φ 값이 1에 가깝게 → 적은 클러스터 생성
- 농도 α를 분산처럼 이해하면 된다

위샤트 분포

- 공분산행렬 샘플링
- 파라미터 d, v가 클러스터 분포 모양 제어

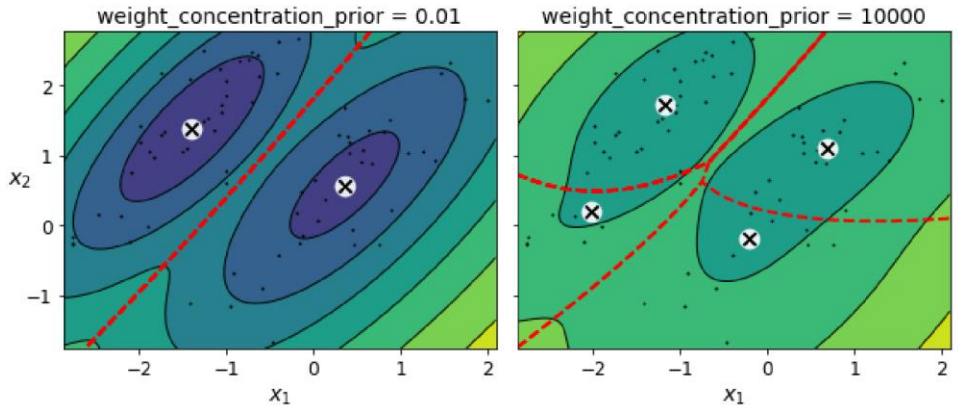


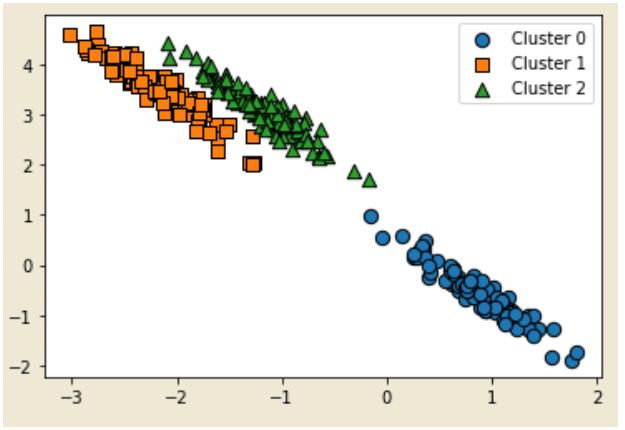
8.1 베이즈가우시안 혼합 모델

사전확률 p(z)

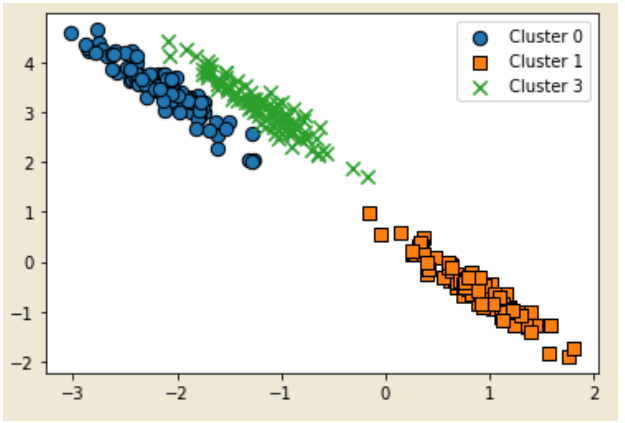
- 잠재변수 z에 대한 사전 지식 (얼마나 확신하느냐)
- 클러스터가 적을 것이라는 사전 지식 = 농도 낮음
- 클러스터가 많을 것이라는 사전 지식 = 농도 높음
- 데이터가 많을수록 사전확률은 별로 중요하지 않음

- weight_concenteration_prior 파라미터로 조정 가능





weight_concen
teration_prior
= 0.01



weight_concen
teration_prior
= 10000

```
np.round(bgm.weights_, 2)
```

```
array([0.33, 0.33, 0.  , 0.33, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  ])
```

8.2 베이즈 정리

베이즈 정리

- 데이터 X를 관측한 후 잠재변수에 대한 확률 분포를 업데이트 하는 방식
- 관측된 데이터 X가 주어졌을 때 z의 조건부 확률 = 사후 확률 분포 계산

$$p(z|X) = \text{사후 확률} = \frac{\text{가능도 x 사전 확률}}{\text{증거}} = \frac{p(X|z)p(z)}{p(X)}$$

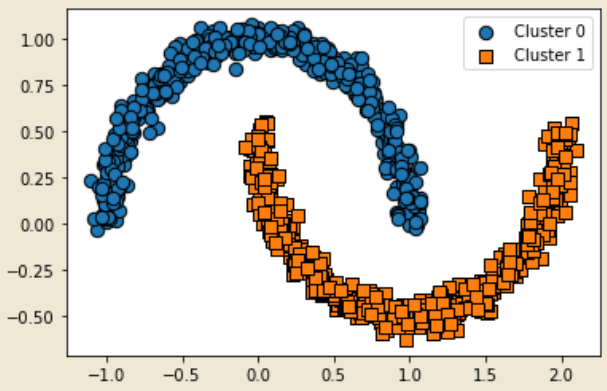
- 증거 $p(X)$는 구하기 어렵기 때문에 변분 추론으로 해결

$$p(X) = \int p(X|z)p(z)dx$$

8.3 타원형 클러스터

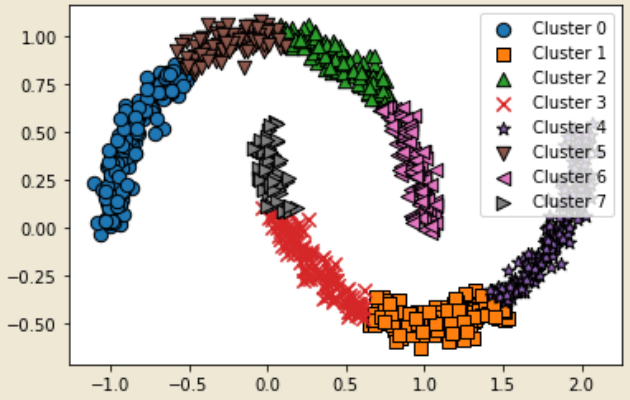
타원형 클러스터에서만 잘 작동
- 반달 데이터셋은 군집화 X
- 이상치 감지에는 활용 가능

```
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=1000, noise=0.05)

# feature 데이터 셋과 make_blobs( ) 의 y 결과 값을 DataFrame으로 저장
clusterDF = pd.DataFrame(data=X, columns=['ftr1', 'ftr2'])
clusterDF['target'] = y
# 생성된 데이터 셋을 target 별로 다른 marker 로 표시하여 시각화 함.
visualize_cluster_plot(None, clusterDF, 'target', iscenter=False)
```

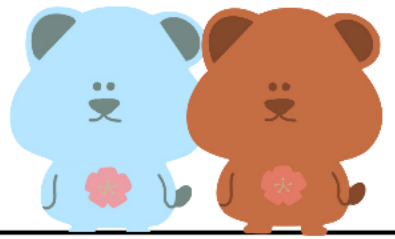



```
bgm = BayesianGaussianMixture(n_components=10, n_init=10, weight_concentration_prior = 0.01, random_state=0)
bgm.fit(X)
bgm_label = bgm.fit(X).predict(X)
clusterDF['bgm_label'] = bgm_label

visualize_cluster_plot(bgm, clusterDF, 'bgm_label',iscenter=False)
```

```
C:\Users\lovel\anaconda3\lib\site-packages\sklearn\mixture\_base.py:265: ConvergenceWarning: Initialization 10
did not converge. Try different init parameters, or increase max_iter, tol or check for degenerate data.
  warnings.warn('Initialization %d did not converge. '
C:\Users\lovel\anaconda3\lib\site-packages\sklearn\mixture\_base.py:265: ConvergenceWarning: Initialization 10
did not converge. Try different init parameters, or increase max_iter, tol or check for degenerate data.
  warnings.warn('Initialization %d did not converge. '
```



THANK YOU



EWHA EURON