

# 11주차 발표

DSE팀 김도하 김경민 냄유럽

# 목차

#01 군집화 개념

#02 K-means

#03 군집평가 성능 지표

#04 평균 이동

#05 병합 군집

#06 GMM

#07 DBSCAN

#08 빠이조가우시안 혼합모델

#09 클러스터 개수 선택





## 01. 군집화 개념

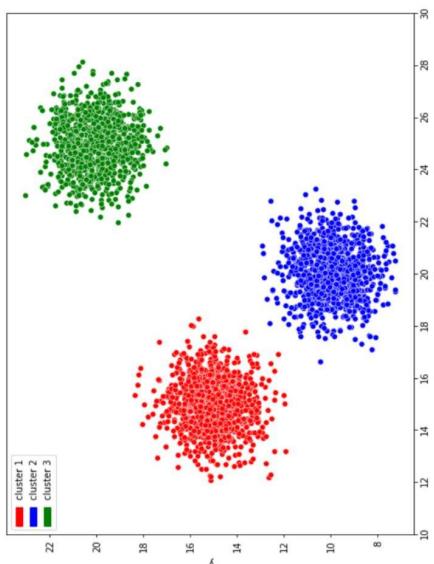
# 1.1 지도학습 VS 비지도학습

지도학습(Supervised Learning)	비지도학습(Unsupervised Learning)
명확한 정답이 주어진 데이터를 먼저 학습한 뒤 미지의 정답을 예측하는 방식	모델에 정답 없이 데이터만 제공한 상태에서 데이터의 패턴을 알아내는 방식
입력값(X data)가 주어지면 입력값에 대한 Label(Y data)를 주어 학습시킴	정답 라벨이 없는 데이터를 비슷한 특징끼리 군집화하여 새로운 데이터에 대한 결과를 예측하는 방법
강아지 사진 X(input data)를 주고 이 사진은 강아지라고 분류해주어 Y값 Label을 알려줌	여러 자동차의 사진을 분류하지 않고 많이 준비한 뒤 자동차 데이터를 머신러닝에게 주어 학습시킴
분류, 회귀, 추천 시스템, 시각/음성 감지/인지, 텍스트 분석, NLP	<b>클러스터링</b> , 차원 축소

# 1.2 군집화 개념

## 군집화

- 비슷한 샘플을 하나의 클러스터로 모으는 것  
ex) 비슷한 행동을 하는 고객을 동이란 클러스터로 분류하는 고객 분류,  
동일한 클러스터 내의 고객들이 좋아하는 컨텐츠를 추천하는 추천 시스템,  
제시된 이미지와 비슷한 이미지를 찾아주는 검색 엔진 등
- 차원 축소를 이용하여 분석을 위한 충분한 정보를 가질 수 있음  
- 이상치 탐지 가능



## 중심 기반 군집화 : K-means

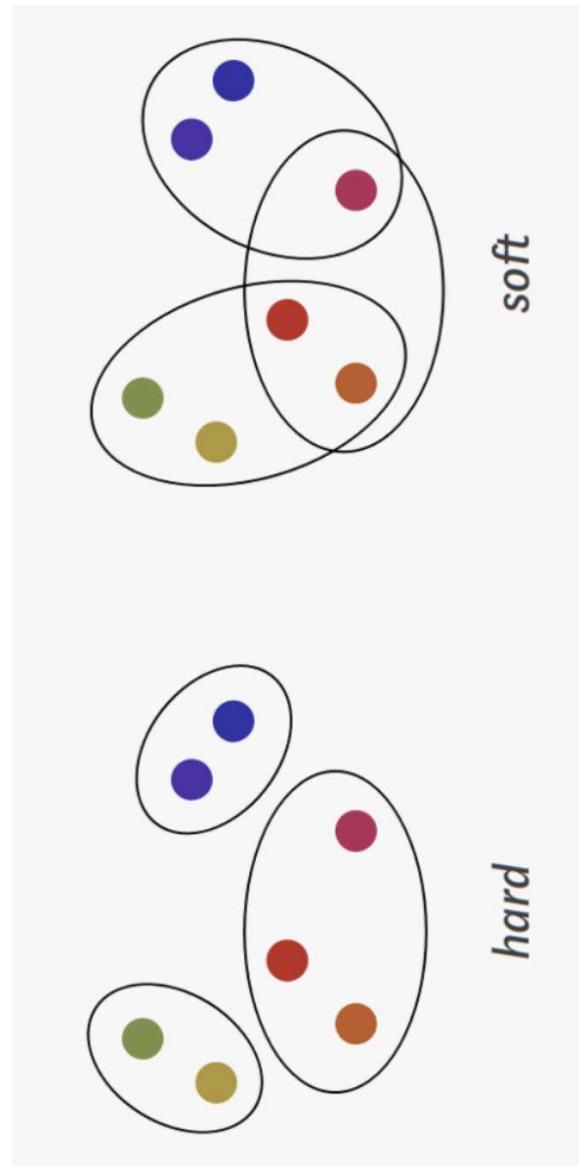
- 비계층적 군집화(Partitioning Clustering)  
: 사전에 군집의 수를 정해주는 방법

## 밀도 기반 군집화 : DBSCAN

- 계층적 군집화(Hierarchical Clustering)  
: 각 개체가 독립적인 각각의 군집에서 점차 거리가 가까운 대상과 군집을 이루는 방법

# 1.3 하드 군집 VS 소프트 군집

하드 군집(클러스터링)	소프트 군집(클러스터링)
하나의 데이터가 정확히 하나의 군집에 할당하는 것 각 샘플에 대해 가장 가까운 cluster를 선택하는 것	하나의 데이터가 다수의 군집에 할당하는 것 Cluster마다 샘플에 점수를 부여하는 것 *점수 : 샘플과 centroid 사이의 거리
hierarchical clustering, k-means, DBSCAN, optics	topic models, fcm, soft k-means





## 02.K-means

## 2.1 군집화 기법 설명 : K-means

- 군집화에서 가장 일반적으로 사용되는 알고리즘
- 군집 중심점(Centroid)이라는 특정한 임의의 지점을 선택해 해당 중심에 가장 가까운 포인트들을 선택하는 군집화 기법
- 선택된 포인트의 평균지점으로 이동하고 이동된 중심점에서 다시 가까운 포인트 선택, 다시 중심점을 평균 지점으로 이동하는 프로세스 반복 수행
- 각 cluster 내 유사도는 높이고 외 유사도는 낮추는 것을 가정으로 각 cluster 거리 차이의 분산을 최대화하는 것이 목적

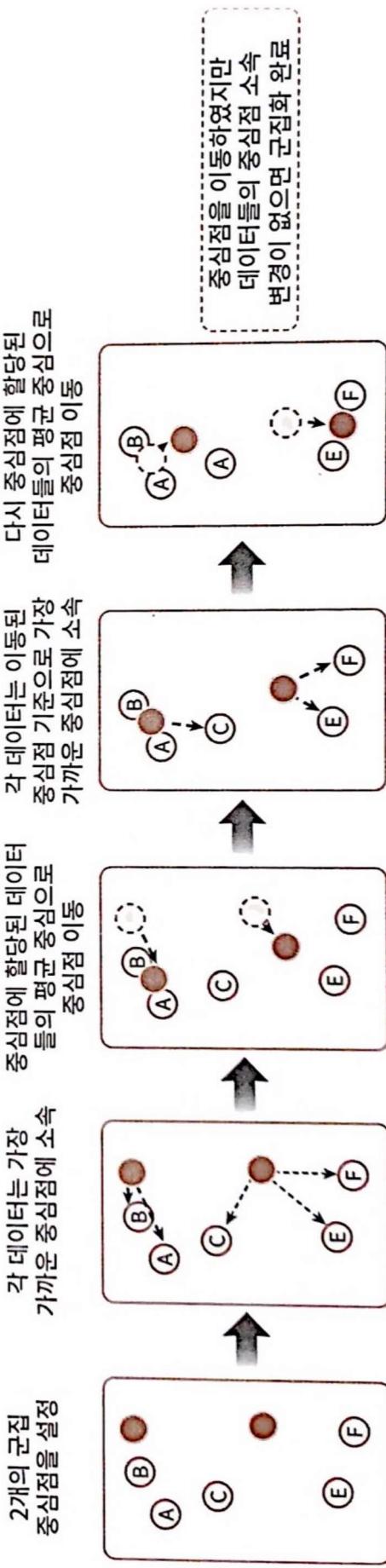
---

### Algorithm 1 $k$ -means algorithm

---

- 1: Specify the number  $k$  of clusters to assign.
  - 2: Randomly initialize  $k$  centroids.
  - 3: repeat
    - 4:     **expectation:** Assign each point to its closest centroid.
    - 5:     **maximization:** Compute the new centroid (mean) of each cluster.
  - 6: until The centroid positions do not change.
-

## 2.2 K-means 수행 과정



A,B,C,D,E는 데이터 포인트이고 ●는 군집 중심점

1. 군집화의 기준이 되는 중심을 구성하려는 군집화 개수만큼 임의의 위치에 가져다 놓음
2. 2개로 군집화하려면 2개의 중심을 임의의 위치에 가져다 놓음
3. 각 데이터는 가장 가까운 곳에 위치한 중심점에 소속됨
4. 이렇게 소속이 결정되면 군집 중심점을 소속된 데이터의 평균 중심으로 이동
5. 중심점이 이동했기 때문에 각 데이터는 기준에 속한 중심점보다 더 가까운 중심점이 있다면 해당 중심점으로 다시 소속 변경
6. 다시 중심을 소속된 데이터의 평균 중심으로 이동
7. 중심점을 이동했는데 데이터의 중심점 소속 변경이 없으면 군집화 종료
8. 그렇지 않다면 다시 4번 과정을 거쳐 소속 변경 & 이 과정 반복

## 2.3 K-means 장단점

장점	단점
일반적인 군집화에서 가장 많이 활용됨 알고리즘이 쉽고 간결	거리 기반 알고리즘 속성의 개수가 매우 많을 경우 군집화의 정확도 떨어짐 (→ PCA 차원 감소를 적용해야 할 수 있음)
	반복을 수행하는데, 반복 횟수가 많을 경우 수행시간이 매우 느려짐
	몇 개의 군집을 선택해야 할지 가이드하기 어렵

## 2.4 K-means의 파라미터

KMeans 클래스의 주요 파라미터	
<b>n_clusters</b>	군집화할 개수, 즉, 군집 중심점의 개수
<b>init</b>	초기에 군집 중심점의 좌표를 설정할 방식 보통 임의로 중심을 설정 X 일반적으로 K-means++방식으로 최초 설정
<b>max_iter</b>	최대 반복 횟수 이 횟수 이전에 모든 데이터의 중심점 이동이 없으면 종료
KMeans 객체의 주요 속성	
<b>labels_</b>	각 데이터 포인트가 속한 군집 중심점 레이블
<b>cluster_centers_</b>	각 군집 중심점 좌표(shape : [군집 개수, 피처개수]) 이를 통해 군집 중심점 좌표가 어디인지 시각화 가능

# 2.5 K-평균을 이용한 봇꽃 데이터 세트 군집화

```

from sklearn.preprocessing import scale
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
%matplotlib inline

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

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

KMeans(n_clusters=3, random_state=0)
  target[] = iris.target
  irisDF['cluster']=kmeans.labels_
  iris_result = irisDF.groupby(['target','cluster'])['sepal_length'].count()
  print(iris_result)

target  cluster
0      1       50
1      0       48
2      2       2
2      0       14
2      2       36
Name: sepal_length, dtype: int64

from sklearn.decomposition import PCA
pca = PCA(n_components=2)
pca_transformed = pca.fit_transform(iris.data)

irisDF[ 'pca_x' ] = pca_transformed[:,0]
irisDF[ 'pca_y' ] = pca_transformed[:,1]
irisDF.head(3)

  sepal_length  sepal_width  petal_length  petal_width  target  cluster  pca_x  pca_y
0         5.1        3.5         1.4        0.2       0       1   -2.684126  0.319397
1         4.9        3.0         1.4        0.2       0       1   -2.714142 -0.177001
2         4.7        3.2         1.3        0.2       0       1   -2.888991 -0.144949

```

# 군집 값이 0, 1, 2인 경우마다 별도의 인덱스로 추출

marker0\_ind = irisDF[irisDF['cluster']==0].index  
marker1\_ind = irisDF[irisDF['cluster']==1].index  
marker2\_ind = irisDF[irisDF['cluster']==2].index

# 군집 값 0, 1, 2에 해당하는 인덱스로 각 군집 레벨의 pca\_x, pca\_y 값 추출. 0, s, ^ 로 마커 표시

plt.scatter(x=irisDF.loc[marker0\_ind, 'pca\_x'], y=irisDF.loc[marker0\_ind, 'pca\_y'], marker='o')  
plt.scatter(x=irisDF.loc[marker1\_ind, 'pca\_x'], y=irisDF.loc[marker1\_ind, 'pca\_y'], marker='s')  
plt.scatter(x=irisDF.loc[marker2\_ind, 'pca\_x'], y=irisDF.loc[marker2\_ind, 'pca\_y'], marker='^')

plt.xlabel('PCA 1')  
plt.ylabel('PCA 2')  
plt.title('3 Clusters Visualization by 2 PCA Components')  
plt.show()

## 2.6 군집화 알고리즘 테스트를 위한 데이터 생성

대표적인 군집화용 데이터 생성기  
여러 개의 클래스에 해당하는 데이터 세트 만들  
하나의 클래스에 여러 개의 군집이 분포될 수 있게 하는 데이터를 생성할 수 있음

**make\_blobs()**  
: 개별 군집의 중심점과 표준편차 제어 가능 추가되어 있음

**make\_classification()**  
: 노이즈를 포함한 데이터를 만드는 데 유용하게 사용할 수 있음

\* make\_circle(), make\_moon() API는 중심 기반의 군집화로 해결하기 어려운 데이터 세트 만드는 데 사용됨

## 2.6 군집화 알고리즘 테스트를 위한 데이터 생성

<code>n_samples</code>	생성할 총 데이터의 개수, 디폴트 = 100
<code>n_features</code>	데이터의 피처 개수, 시각화를 목표로 할 경우 2개로 설정, 보통 첫번째 피쳐는 x 좌표, 두번째 피쳐는 y 좌표
<code>centers</code>	int 값, 3이라면 군집의 개수를 나타냄 <code>ndarray</code> 형태로 표현할 경우 개별 군집 중심점의 좌표 의미
<code>cluster_std</code>	생성될 군집 데이터의 표준 편차 의미 float 값 0.8과 같은 형태로 지정) 군집 내에서 데이터가 표준편차 0.8을 기준 값으로 만들어짐 [0.8, 1.2, 0.6]과 같은 형태로 지정) 첫번째 군집 내 데이터의 표준편차 0.8, 두번째 . . . , 세번째 . . . -> 군집별로 서로 다른 표준편차를 가진 데이터 세트를 만들 때 사용

## 2.6 군집화 알고리즘 테스트를 위한 데이터 생성

```

# KMeans 객체를 이용하여 X 데이터를 K-Means 클러스터링 수행
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=200, random_state=0)
cluster_labels = kmeans.fit_predict(X)
clusterDF['kmeans_label'] = cluster_labels

X, y = make_blobs(n_samples=200, n_features=2, centers=3, cluster_std=0.8, random_state=0) #cluster_centers_는 개별 클러스터의 중심 위치 좌표 시각화를 위해 추출
centers = kmeans.cluster_centers_
unique_labels = np.unique(cluster_labels)
unique_counts = np.unique(y, return_counts=True)
print(unique_counts)

(200, 2) (200, )
[0 1 2] [67 67 66]

import pandas as pd
clusterDF = pd.DataFrame(data=X, columns=['ftr1', 'ftr2'])
clusterDF['target'] = y
clusterDF.head(3)

ftr1      ftr2      target
0 -1.692427  3.622025      2
1  0.697940  4.428867      0
2  1.102228  4.806317      0

target_list = np.unique(y)
# 각 터미널 단계의 마커 지정,
markers=[ 'o', '^', 's', 'v', 'P', 'D', 'H', 'x' ]
# 3개의 군집 영역으로 구분한 데이터 세트를 생성함으로서 target_list=[0, 1, 2]
# target==0, target==1, target==2로 scatter plot을 marker별로 생성.
for target in target_list:
    target_cluster = clusterDF[clusterDF['target']==target]
    plt.scatter(x=target_cluster['ftr1'], y=target_cluster['ftr2'], edgecolor='k',
                marker=markers[target])
plt.show()

# 군집별 중심 위치 좌표 시각화
plt.scatter(x=center_X_y[0], y=center_X_y[1], s=200, color='white',
            alpha=0.9, edgecolor='k', marker=markers[0])
plt.scatter(x=center_X_y[0], y=center_X_y[1], s=70, color='k', edgecolor='k',
            marker='^' % label)

plt.show()

```

```

print(clusterDF.groupby('target')[['kmeans_label']].value_counts())

```

target	kmeans_label	value_counts
0	0	66
1	1	1
2	2	67
	1	65
	2	1

```

Name: kmeans_label, dtype: int64

```



## 03. 구집평가

# 3.1 실루엣 분석(Silhouette Analysis)

## 실루엣 분석(Silhouette Analysis)

- 각 군집 간의 거리가 얼마나 효율적으로 분리돼 있는지
  - 효율적으로 잘 분리되었다 = 다른 군집과의 거리는 떨어져 있고 동일 군집끼리의 데이터는 서로 가깝게 잘 뭉쳐있다
  - 군집화가 잘 될수록 개별 군집은 비슷한 정도의 여유공간을 가지고 떨어져 있을 것
  - 실루엣 계수(Silhouette Coefficient) 기반
- 실루엣 계수 : 개별 데이터가 가지는 군집화 지표  
 개별 데이터가 가지는 실루엣 계수는 해당 데이터와 얼마나 가깝게 군집화돼 있고,  
 다른 군집에 있는 데이터와는 얼마나 멀리 분리돼 있는지를 나타내는 지표

## 실루엣 계수 계산

특정 데이터 포인트의 실루엣 계수 값 :  $a(i), b(i)$  기반으로 계산

\*  $a(i)$  : 해당 데이터 포인트와 같은 군집 내에 있는 다른 데이터 포인트와의 거리를 평균한 값

\*  $b(i)$  : 해당 데이터 포인트가 속하지 않은 군집 중 가장 가까운 군집과의 평균 거리

두 군집 간의 거리가 얼마나 떨어져 있는가의 값은  $b(i) - a(i)$

이 값을 정규화하기 위해  $\text{MAX}(a(i), b(i))$  값으로 나눔

$$\therefore i \text{ 번째 데이터 포인트의 실루엣 계수 값 } s(i) = \frac{(b(i) - a(i))}{\text{MAX}(a(i), b(i))}$$

실루엣 계수 : -1과 1사이

1로 가까워질수록 근처의 군집과 더 멀리 떨어져있다. 0에 가까울수록 근처의 군집과 가까워져있다.

- 값은 아래 다른 군집에 데이터 포인트가 할당되었다.

## 3.2 실루엣 분석을 위한 데서드

`sklearn.metrics.silhouette_samples(X, labels, metric = 'euclidean', **kwds)`  
인자로 X feature 데이터 세트와 각 피처 데이터가 속한 군집 레이블 값인 labels 데이터 입력  
⇒ 각 데이터 포인트의 실루엣 계수를 계산해 반환

`sklearn.metrics.silhouette_score(X, labels, metric = 'euclidean', sample_size = None, **kwds)`  
인자로 X feature 데이터 세트와 각 피처 데이터가 속한 군집 레이블 값인 labels 데이터 입력  
⇒ 전체 데이터의 실루엣 계수 값을 평균해 반환. 즉 np.mean(silhouette\_samples())  
일반적으로 이 값이 높을수록 군집화가 어느정도 잘 되었다고 판단 가능  
하지만 무조건 이 값이 높다고 해서 군집화가 잘 되었다고 판단할 수 없음

### 3.3 좋은 군집화

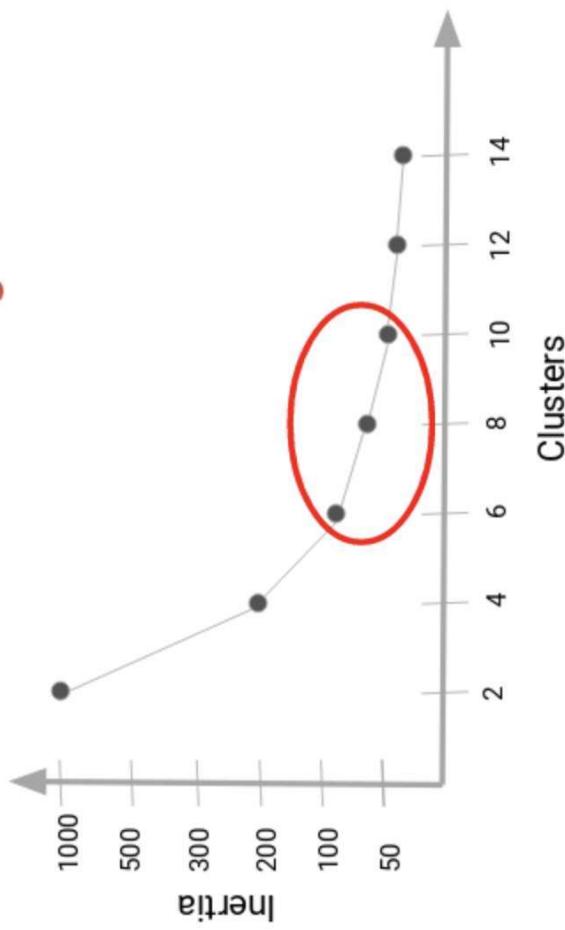
좋은 군집화가 되려면

1. 전체 실루엣 계수의 평균값, 즉 사이킷런의 silhouette\_scores() 값은 0 ~ 1 사이의 값을 가지며 1에 가까울수록 좋음
2. 전체 실루엣 계수의 평균값과 더불어 개별 군집의 평균값의 편차가 크지 않아야 함  
즉, 개별 군집의 실루엣 계수 평균값이 전체 실루엣 계수의 평균값에서 크게 벗어나지 않는 것 중요  
만약 전체 실루엣 계수의 평균값은 높지만, 특정 군집의 실루엣 계수 평균값만 유난히 높고  
다른 군집들의 실루엣 계수 평균값은 낮으면 중요 군집화 조건이 아님

## 3.4 Inertia, Elbow

### 이너시(Inertia)

- 각 샘플과 가장 가까운 중심점 사이의 평균 제곱 거리를 측정한 수치
- 클러스터 수와 이너시는 반비례 관계
- 좋은 모델을 만들기 위해서 이너시를 성능 지표로 선택하는 것 좋지 않음
- 이너시를 줄이기 위해서 클러스터의 수를 실제 훈련 세트에 있는 클러스터 수보다 많이 생성할 것이기 때문  
→ 클러스터 수 ↑, 이너시 급격하게 ↓, 어느 지점에서는 완만하게 감소하는데  
이 지점을 엘보우(Elbow)라고 하는데, 이를 모델의 적정 클러스터 수로 지정할 수 있다.



# 3.5 불꽃 데이타 세트를 이용한 군집 풍자

```

from sklearn.preprocessing import scale
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
# 실루엣 분석 metric 값을 추가한 API 추가
from sklearn.metrics import silhouette_samples, silhouette_score
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

%matplotlib inline

iris = load_iris()
feature_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
irisDF = pd.DataFrame(data=iris.data, columns=feature_names)
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, random_state=0).fit(irisDF)

irisDF['cluster'] = kmeans.labels_

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

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

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

irisDF.head(3)

silhouette_samples( ) return 값의 shape (150, )
불꽃 데이타셋 Silhouette Analysis Score:0.553
   sepal_length  sepal_width  petal_length  petal_width  cluster  silhouette_coeff
0          5.1         3.5         1.4         0.2         1        0.852955
1          4.9         3.0         1.4         0.2         1        0.815495
2          4.7         3.2         1.3         0.2         1        0.829315

```

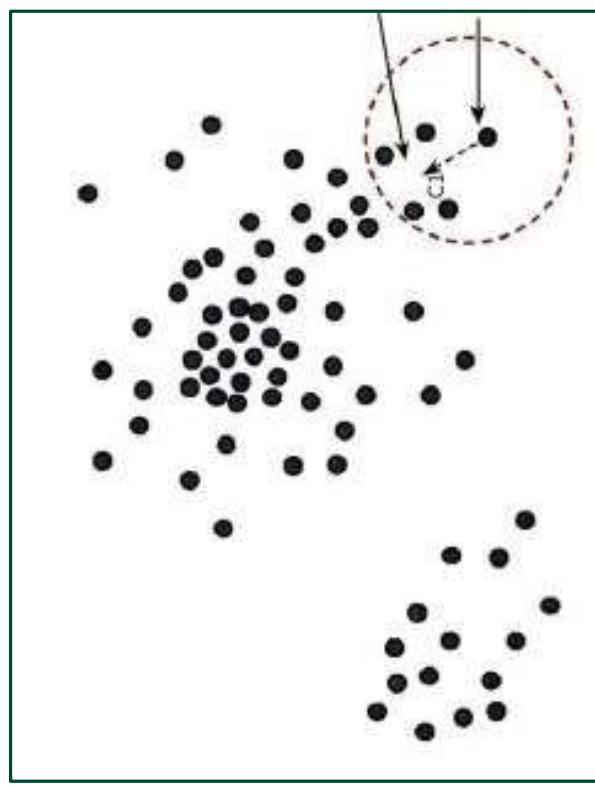


#04 펑크 이동

# #04 평균 이동

## 1. 평균 이동(Mean Shift)의 개요

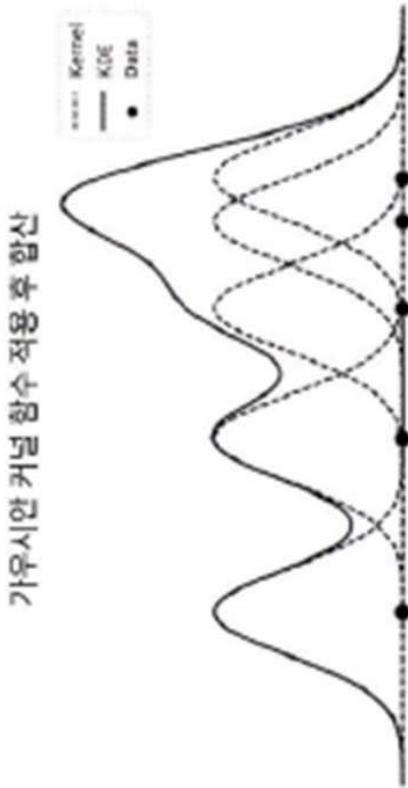
- 중심을 군집의 중심으로 지속적으로 움직이면서 군집화 수행 (cf. K-평균과 유사)
- 데이터가 모여 있는 밀도가 가장 높은 곳으로 이동 (cf. K-평균은 데이터의 평균 거리 중심으로 이동)
- 데이터의 분포도로 군집 중심점 탐색 -> 확률 밀도 함수 -> KDE
  - 1) 개별 데이터의 특정 반경 내에 주변 데이터를 포함한 데이터 분포도를 KDE 기반 Mean Shift 알고리즘으로 계산
  - 2) 데이터 분포도가 높은 방향으로 데이터 이동
  - 3) 모든 데이터를 1), 2)를 수행하면서 이동하면 개별 데이터들이 군집 중심점으로 모임
  - 4) 지정된 반복 횟수만큼 전체 데이터에 대해 KDE 기반 군집화 수행
  - 5) 개별 데이터들이 모인 중심점을 군집 중심점으로 설정



# #04 평균 0|동

## 2. KDE(Kernel Density Estimation)

- 확률 밀도 함수(Probability Density Function, PDF) : 확률 변수의 분포를 나타내는 함수
- KDE : 커널 함수를 통해 어떤 변수의 확률 밀도 함수를 추정하는 대표적인 방법
- 개별 관측 데이터에 커널 함수를 적용한 값을 모두 더한 후 데이터 건수로 나누 확률 밀도 함수 추정
- 대표적인 커널 함수로 가우시안(정규분포) 함수 사용



# #04 평균 이동

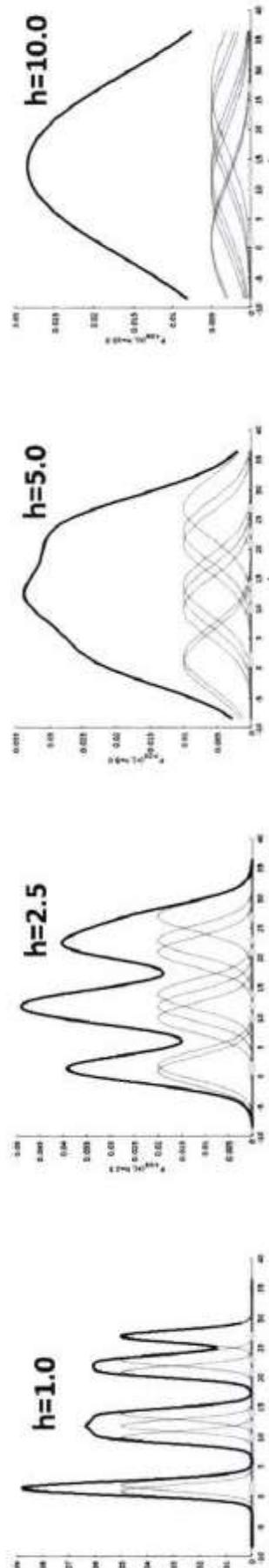
## 2. KDE(kernel Density Estimation)

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

K : 커널함수   x : 확률 변수의 값    $x_i$  : 관측값   h : 대역폭

- 대역폭(bandwidth) : KDE 형태를 부드러운 형태로 평활화하는 데 적용 -> 확률 밀도 추정 성능 크게 좌우

- 작은  $h$ 값 -> 좁고 뾰족한 KDE -> 변동성이 큰 방식으로 추정 -> 과적합 가능성
- 큰  $h$ 값 -> 과도하게 평활화된 KDE -> 지나치게 단순화된 방식으로 추정 -> 과소적합 가능성



# #04 평균 이동

## 3. 사이킷런 MeanShift 클래스

```
import numpy as np
from sklearn.datasets import make_blobs 군집화 알고리즘 테스트를 위한 데이터 생성
from sklearn.cluster import MeanShift 사이킷런 MeanShift 임포트

X, y = make_blobs(n_samples=200, n_features=2, centers=3, cluster_std=0.7, random_state=0)
 표준편차가 0.7인 3개 군집의 데이터 생성

meanshift = MeanShift(bandwidth=0.8) KDE의 대역폭 h와 동일한 의미를 가지는 주요 파라미터
cluster_labels = meanshift.fit_predict(X)
print('cluster_labels 유형 : ', np.unique(cluster_labels))

cluster_labels 유형 : [0 1 2 3 4 5]

meanshift = MeanShift(bandwidth=1.0)
cluster_labels = meanshift.fit_predict(X)
print('cluster_labels 유형 : ', np.unique(cluster_labels))

cluster_labels 유형 : [0 1 2]
bandwidth 값 : 1.816

Estimate_bandwidth()
• 최적화된 bandwidth를 찾기 위한 함수
• 파라미터로 피처 데이터 세트 입력
```

# #04 평균 이동

## 3. 사이킷런 MeanShift 클래스

```
import pandas as pd
clusterDF = pd.DataFrame(data=X, columns=['ftr1', 'ftr2'])
clusterDF['target'] = y

best_bandwidth = estimate_bandwidth(X)

meanshift = MeanShift(bandwidth=best_bandwidth)

cluster_labels = meanshift.fit_predict(X)
print('cluster labels 유형 : ', np.unique(cluster_labels))

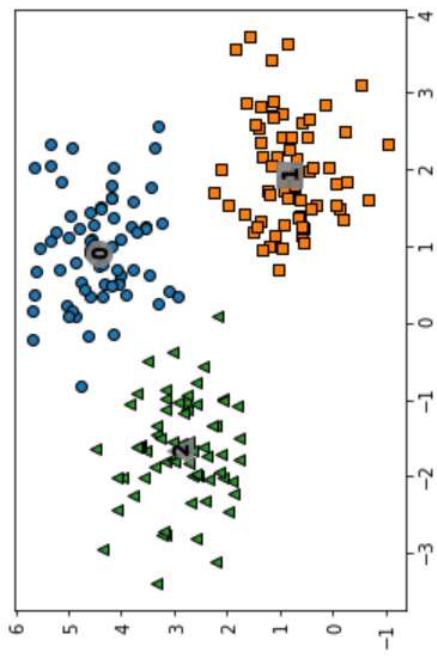
cluster_labels 유형 : [0 1 2]
```

```
import matplotlib.pyplot as plt
%matplotlib lib inline

clusterDF['meanshift_label'] = cluster_labels
centers = meanshift.cluster_centers_
unique_labels = np.unique(cluster_labels)
markers=['o', 's', '^', 'x', '*']

for label in unique_labels :
    label_ccluster = clusterDF[clusterDF['meanshift_label'] == label]
    center_x,y = centers[[label]]
    plt.scatter(x=label_ccluster['ftr1'], y=label_ccluster['ftr2'], edgecolor='k', marker=markers[[label]])
    plt.scatter(x=center_x,y[0], y=center_y[0], s=200, color='gray', alpha=0.9, marker=markers[[label]])
    plt.scatter(x=center_x,y[0], y=center_y[1], s=70, color='k', edgecolor='k', marker=markers[[label]])
plt.show()
```

```
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
```



# #04 평균 이동

## 4. 평균 이동 장단점

### 장점

- 데이터 세트 형태를 특정 형태로 가정하거나, 특정 분포도 기반 모델로 가정하지 않음 -> 유연한 군집화
  - 이상치의 영향력이 크지 않음
  - 미리 군집 개수를 정할 필요 없음

### 단점

- 알고리즘 수행 시간이 오래 걸림
- 대역폭에 따라 군집화 성능이 좌우됨



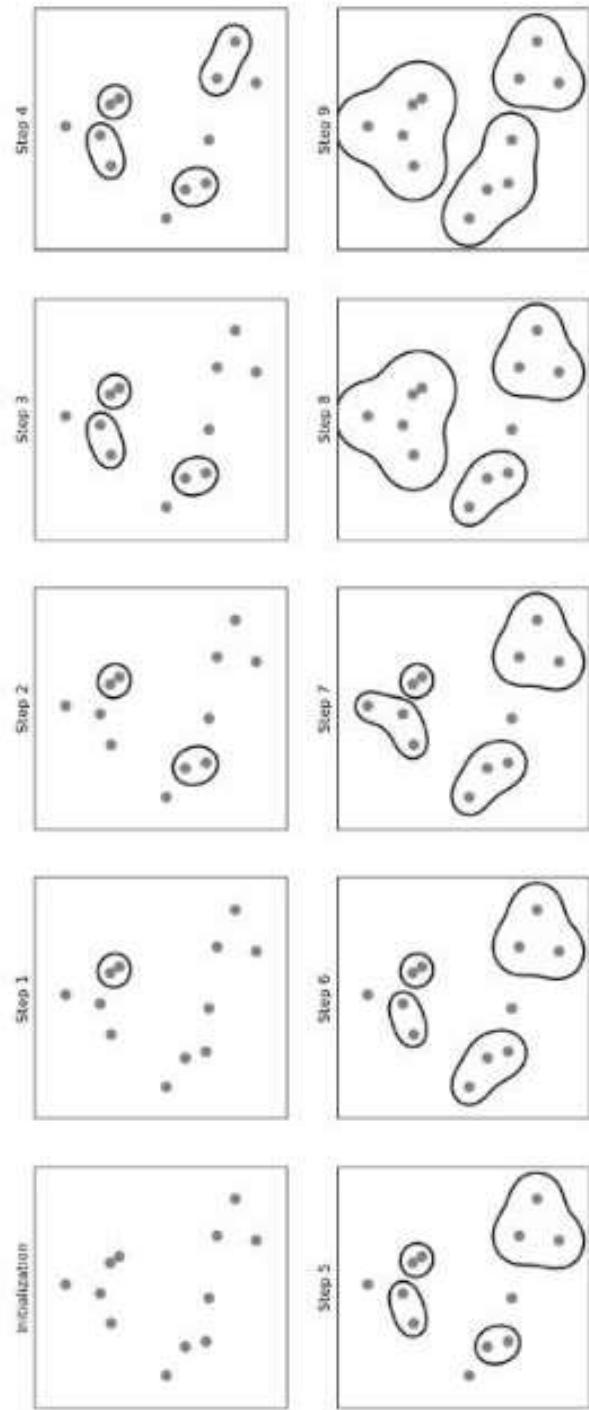
## #05 병합 구조

# #05 병합 군집

<https://kolikim.tistory.com/31>

## 1. 병합 군집(Agglomerative Clustering)

- 각각의 데이터 포인트를 하나의 클러스터로 지정하고, 지정된 개수의 클러스터가 남을 때까지 가장 비슷한 두 클러스터를 합쳐 나가는 알고리즘
- 작은 클러스터들이 모여 큰 클러스터를 이루는 계층적 구조

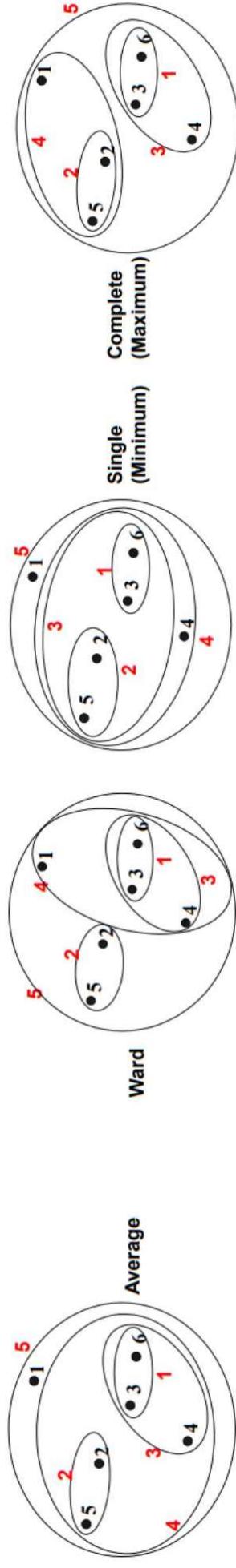


# #05 병합 군집

<https://bizzengine.tistory.com/152>

## 1. 병합 군집(Agglomerative Clustering)

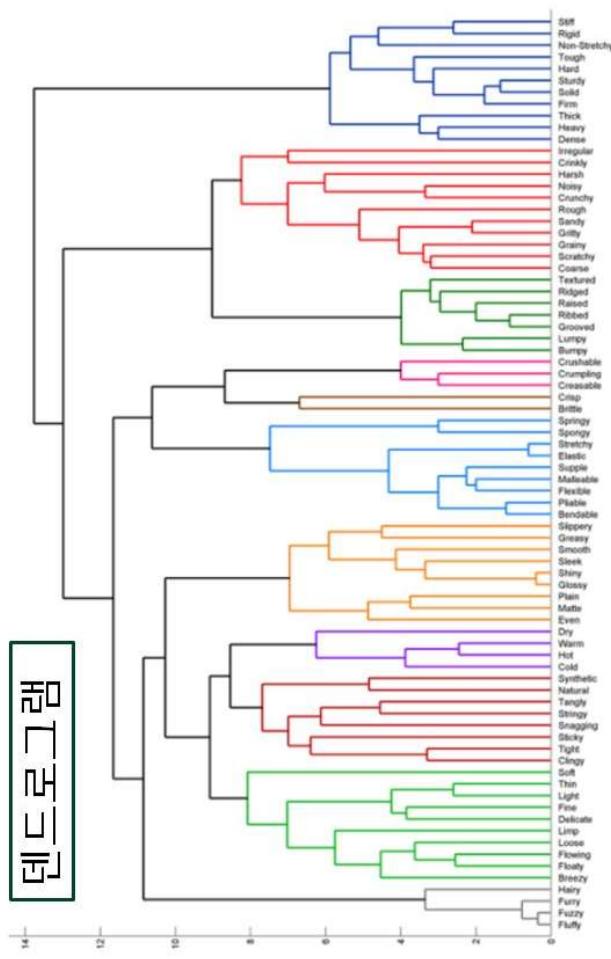
- Linkage 옵션 -> 클러스터를 합쳐나가는 방식
- Ward : 모든 클러스터 내의 분산을 가장 작에 증가시키는 두 클러스터를 합치는 방식
- Average : 포인트 사이의 평균 거리가 가장 짧은 두 클러스터를 합치는 방식
- Complete : 포인트 사이의 최대 거리가 가장 짧은 두 클러스터를 합치는 방식
- Single : 포인트 사이의 최소 거리가 가장 짧은 두 클러스터를 합치는 방식



## #05 병합 군집

## 2. 계층적 군집(Hierarchical Clustering)

- 계층적 트리 모형을 이용하여 군집화 수행
- 개별 데이터 포인트의 순차적, 계층적으로 유사한 클러스터로 통합
- 병합 군집(상향식) vs 분할 군집(하향식)



# #05 병합 군집

## 3. 사이킷런 AgglomerativeClustering 클래스

<https://data-newbie.tistory.com/25>

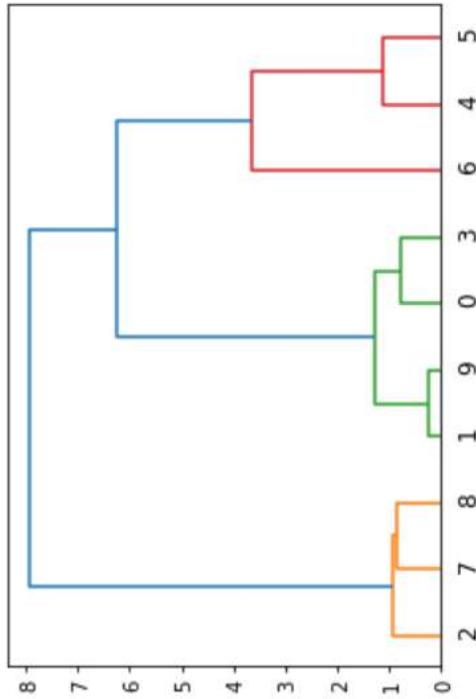
```
from sklearn.cluster import AgglomerativeClustering
from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=10, n_features=2, centers=3, cluster_std=0.7, random_state=0)

agg = AgglomerativeClustering(n_clusters=3, linkage="ward")
cluster_labels = agg.fit_predict(X)
print('cluster labels 유형 : ', np.unique(cluster_labels))

cluster_labels 유형 : [0 1 2]
```

```
from scipy.cluster.hierarchy import dendrogram, ward
linkage_array = ward(X)
dendrogram(linkage_array)
```



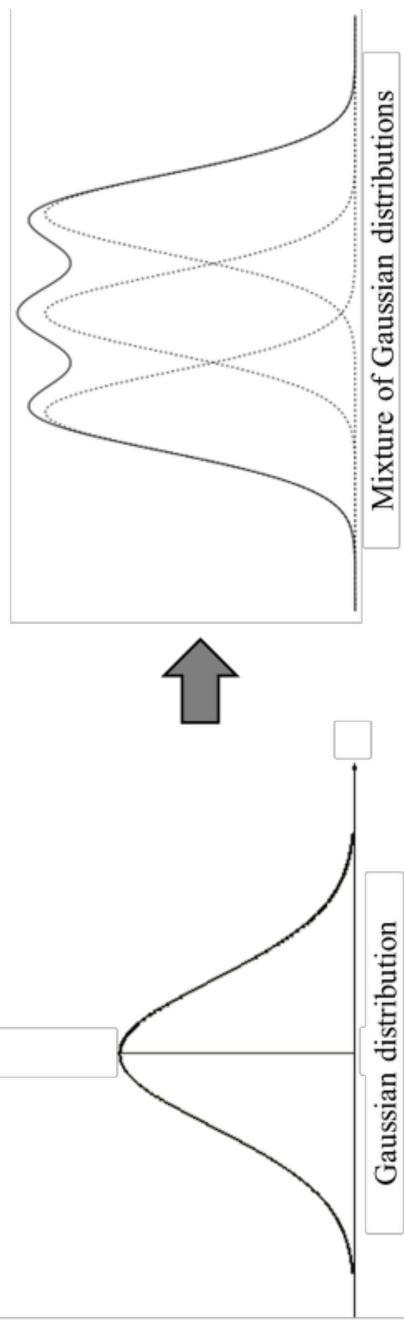


#06 GMM

## #06 GMM

## 1. GMM(Gaussian Mixture Model)

- 데이터가 여러 개의 가우시안 분포를 가진 데이터 집합들이 섞여서 생성된 것이라는 가정하에 군집화 수행
- 섞인 데이터 분포에서 개별 유형의 가우시안 추출 -> 개별 데이터가 이 중 어떤 정규 분포에 속하는지 결정
- 모수 추정 : 개별 정규 분포의 평균과 분산 & 각 데이터가 어떤 정규 분포에 해당되는지의 확률



## #06 GMM

## 2. 사이킷런 GaussianMixture 클래스

## [붓꽃 데이터 세트]

```
from sklearn.datasets import load_iris
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
%matplotlib inline
```

```
iris = load_iris()
feature_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']

irisDF = pd.DataFrame(data=iris.data, columns=feature_names)
irisDF['target'] = iris.target
```

## [GMM 주요 파라미터]

`n_components :`

가우시안 혼합 모델의 총 개수, 군집 개수 설정에

```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3, random_state=0).fit(iris.data)
gmm_labels = gmm.predict(iris.data)

irisDF['gmm_cluster'] = gmm_labels
irisDF['target'] = iris.target

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

target	gmm_cluster
0	0
1	2
2	1

Name: gmm\_cluster, dtype: int64

## #06 GMM

## 3. GMM vs K-Means

```

from sklearn.datasets import make_blobs

X, y = make_blobs(n_samples=300, n_features=2, centers=3, cluster_std=0.5, random_state=0)
transformation = [[0.60834549, -0.63667341], [-0.40887718, 0.85253229]]
```

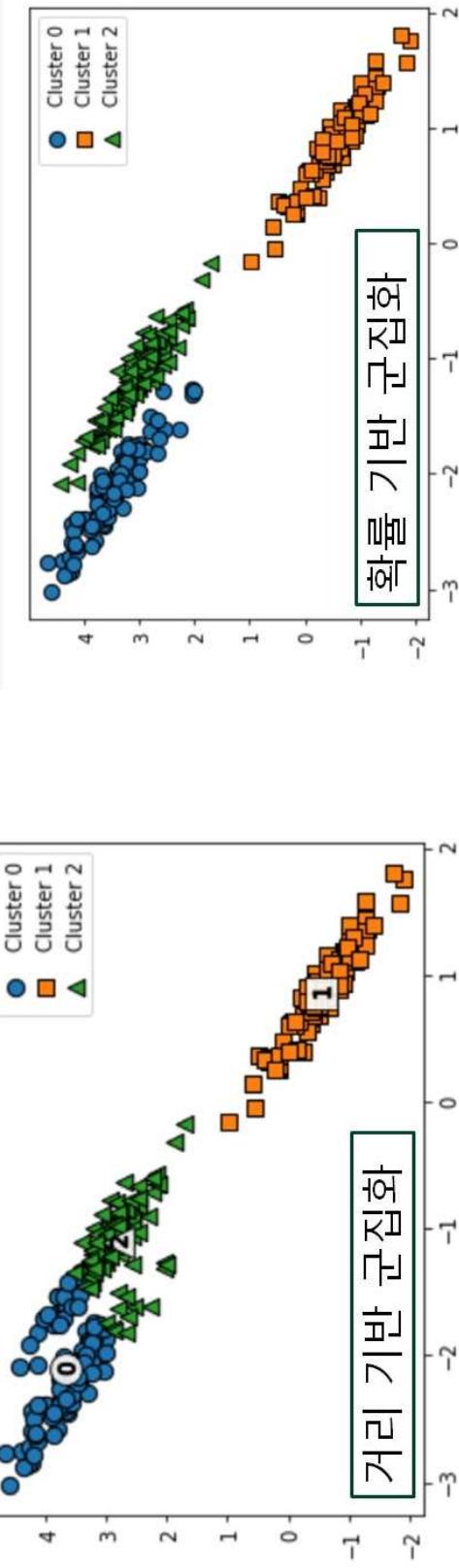
```

kmeans = KMeans(3, random_state=0)
kmeans_label = kmeans.fit_predict(X_aniso)
clusterDF['kmeans_label'] = kmeans_label

visualize_cluster_plot(kmeans, clusterDF, 'kmeans_label', iscenter=True)

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

```



# #06 GMM

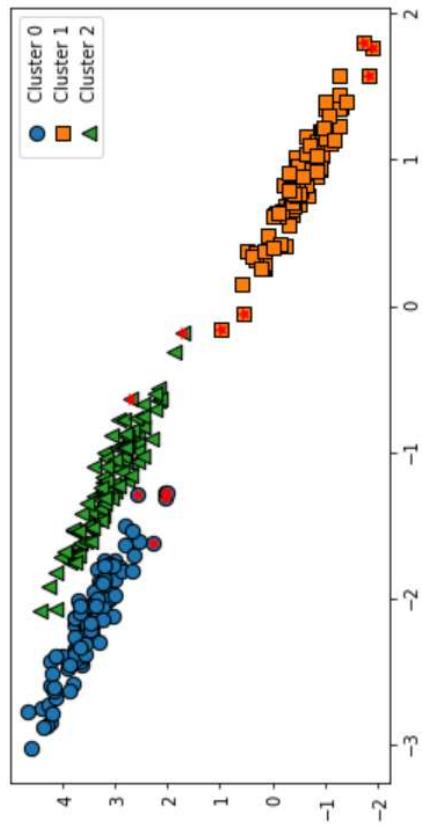
<https://berrrrr.github.io/datascience/2020/11/16/hands-on-ml-9/>

## 4. 이상치 탐지(Outlier Detection)

- 보통과 많이 다른 샘플을 감지하는 작업
- 밀도가 낮은 지역에 있는 모든 샘플을 이상치로 보고 탐지 가능
- 밀도 임계값을 설정하고, 임계값이 넘는 데이터를 Outlier로 판단

```
densities = gmm.score_samples(X_aniso)
density_threshold = np.percentile(densities, 4)
anomalies = X_aniso[densities < density_threshold]

plt.figure(figsize=(8,4))
visualize_cluster_plot(gmm, clusterDF, 'gmm_label', iscenter=False)
plt.scatter(anomalies[:, 0], anomalies[:, 1], color='r', marker='*')
plt.show()
```

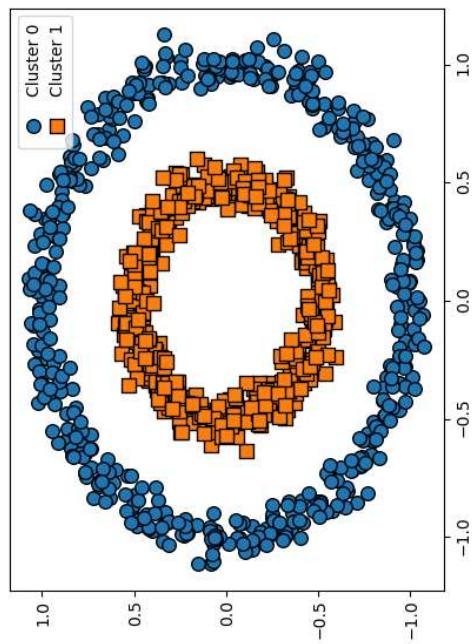




## 7. DBSCAN

# 7.1 DBSCAN 개요

- 간단하고 직관적인 알고리즘으로 돼있음에도 데이터의 분포가 기하학적으로 복잡한 데이터 세트에도 효과적인 군집화가 가능함.



- 특정 공간 내에 데이터 밀도 차이를 기반 알고리즘으로 하고 있어서 복잡한 기하학적 분포도를 가진 데이터 세트에 대해서도 군집화를 잘 수행함.

## 7.2 중요 파라미터 & 데이터 포인트

### [중요 파라미터]

입실론 주변 영역(epsilon) : 개별 데이터를 중심으로 입실론 반경을 가지는 원형의 영역

최소 데이터 개수(min points) : 개별 데이터의 입실론 주변 영역에 포함되는 타 데이터의 개수

### [데이터 포인트]

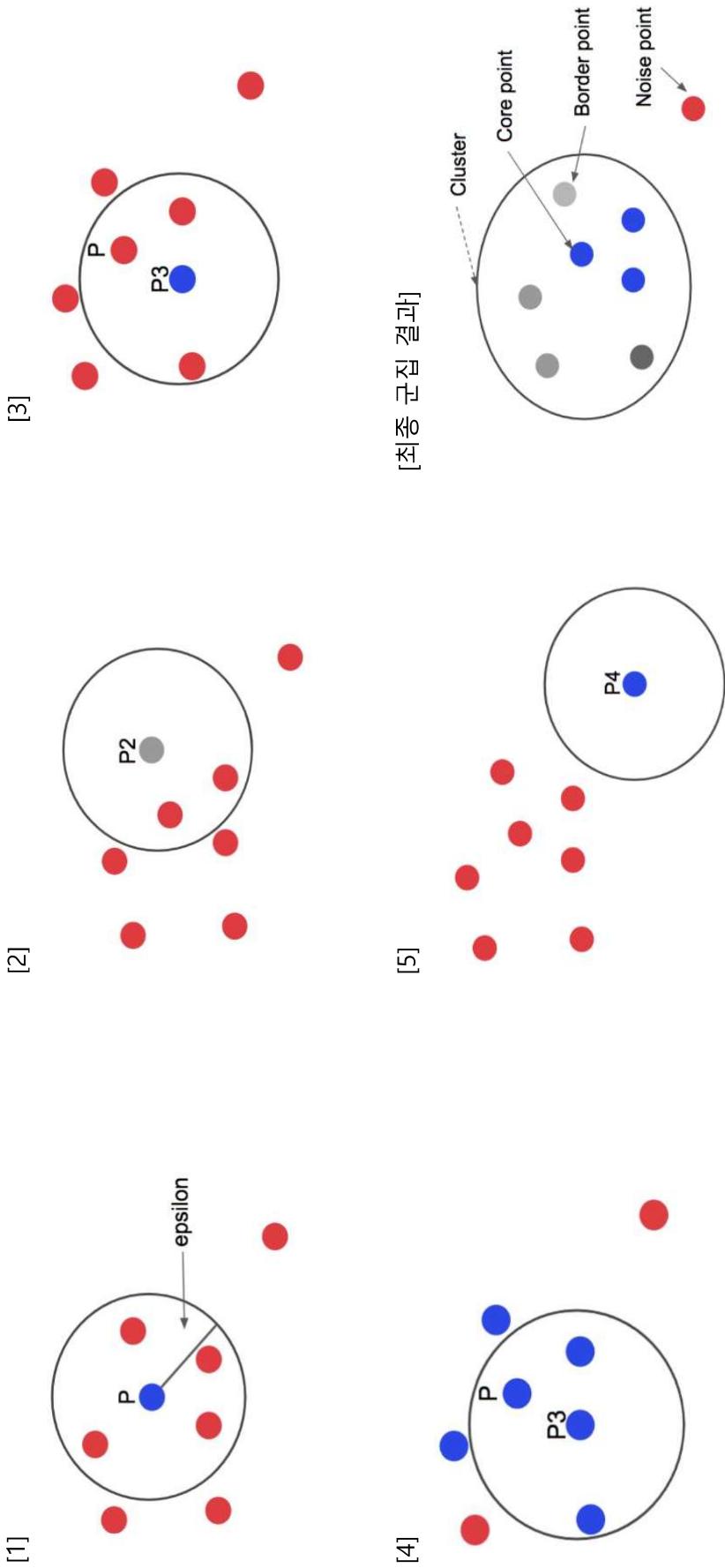
핵심 포인트(Core Point) : 주변 영역 내에 최소 데이터 개수 이상의 타 데이터를 가지고 있을 경우

이웃 포인트(Neighbor Point) : 주변 영역 내에 위치한 타 데이터

경계 포인트(Border Point) : 주변 영역 내에 최소 데이터 개수 이상의 이웃 포인트를 가지고 있지 않지만 핵심 포인트를 이웃 포인트로 가지고 있는 데이터

잡음 포인트(Noise Point) : 최소 데이터 개수 이상의 이웃 포인트를 가지고 있지 않으며, 핵심 포인트도 이웃 포인트로 가지고 있지 않는 데이터

## 7.2 중요 파라미터 & 데이터 포인트



DBSCAN : 입실론 주변 영역의 최소 데이터 개수를 포함하는 밀도 기준을 충족시키는 데이터인 핵심 포인트를 연결하면서 군집화를 구성하는 방식

출처 : 클러스터링 DBSCAN(밀도 기반 클러스터링)(tistory.com)

# 7.3 DBSCAN 적용하기(붓꽃)

## (1) 데이터 로드

```

1 from sklearn.datasets import load_iris
2
3 import matplotlib.pyplot as plt
4 import numpy as np
5 import pandas as pd
6 %matplotlib inline
7
8 iris = load_iris()
9 feature_names = ['sepal_length', 'sepal_width', 'petal_length', 'petal_width']
10 # 보다 편리한 대입법 Handing을 위해 DataFrame으로 변환
11 irisDF = pd.DataFrame(iris.data, columns=feature_names)
12 irisDF['target'] = iris.target
13

```

## (3) PCA 이용해 2개의 피처로 압축 변환 후 시각화

```

1 from sklearn.decomposition import PCA
2 # 2개의 피처로 시각화하기 위해 PCA n_components=2로 적용(데이터 세트 때문)
3 pca = PCA(n_components=2, random_state=0)
4 pca_transformed = pca.fit_transform(iris.data)
5 # visualize clustering 결과를 표시할 때 PCA 변환값을 해당 클러스터로 생성
6 irisDF[['ftr1']] = pca_transformed[:, 0]
7 irisDF[['ftr2']] = pca_transformed[:, 1]
8 visualize_cluster_plot(dbSCAN, irisDF, 'dbSCAN', 'iscenter=False')

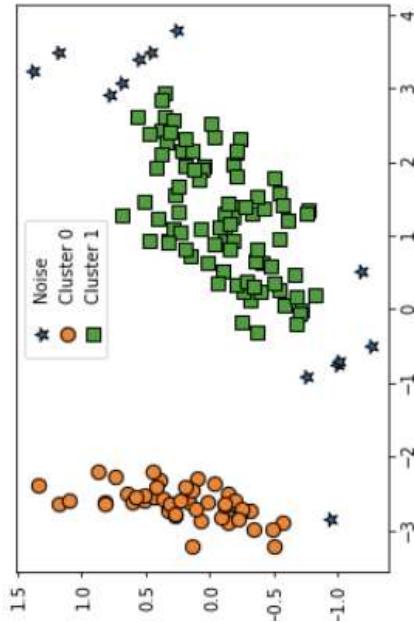
```

## (2) (eps = 0.6, min\_samples=8)로 군집화

```

1 from sklearn.cluster import DBSCAN
2 dbSCAN = DBSCAN(eps=0.6, min_samples=8, metric='euclidean')
3 dbSCAN.labels_ = dbSCAN.fit_predict(iris.data)
4
5 irisDF['dbSCAN_cluster'] = dbSCAN.labels_
6 irisDF['target'] = iris.target
7
8 iris_result = irisDF.groupby(['target'])['dbSCAN_cluster'].value_counts()
9
10 print(iris_result)

```



# 7.3 DBSCAN 적용하기(붓꽃)

(4) (eps = 0.8, min\_samples=8)로 군집화

```

1 from sklearn.cluster import DBSCAN
2
3 dbSCAN = DBSCAN(eps=0.8, min_samples=8, metric='euclidean')
4 dbSCAN_labels = dbSCAN.fit_predict(iris.data)
5
6 irisDF['dbSCAN_cluster'] = dbSCAN_labels
7 irisDF['target'] = Iris.target
8
9 iris_result = irisDF.groupby(['target']).value_counts()
10 print(iris_result)
11
12 visualize_cluster_plot(dbSCAN, irisDF, 'dbSCAN_cluster', iscenter=False)
13
14 target dbSCAN_cluster
15      0          0        48
16      1          1        2
17      2          1        44
18      -1         2        6
19
20 Name: dbSCAN_cluster, dtype: int64

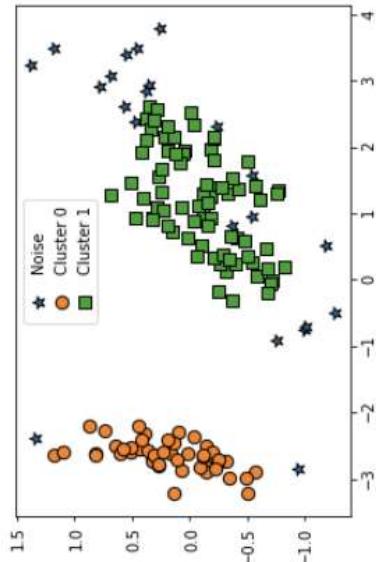
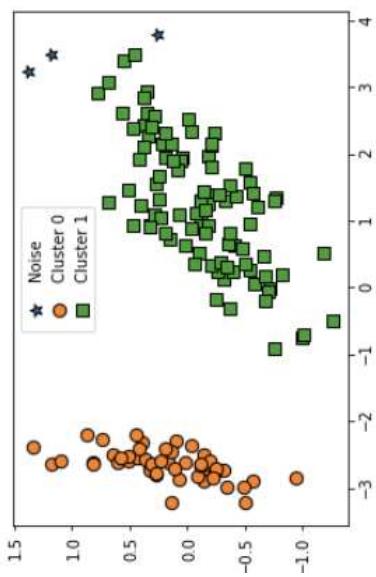
```

(5) (eps = 0.6, min\_samples=16)로 군집화

```

1
2 dbSCAN = DBSCAN(eps=0.6, min_samples=16, metric='euclidean')
3 dbSCAN_labels = dbSCAN.fit_predict(iris.data)
4
5 irisDF['dbSCAN_cluster'] = dbSCAN_labels
6 irisDF['target'] = Iris.target
7
8 iris_result = irisDF.groupby(['target']).value_counts()
9
10 print(iris_result)
11
12 visualize_cluster_plot(dbSCAN, irisDF, 'dbSCAN_cluster', iscenter=False)
13
14 target dbSCAN_cluster
15      0          0        48
16      1          1        2
17      2          1        36
18      -1         2        14
19
20 Name: dbSCAN_cluster, dtype: int64

```



# 7.3 DBSCAN 적용하기(make\_circles())

## (1) make\_circles() 힘수

```
# KMeans은 make_circles() 데이터 세트를 클러스터링 수령.
from sklearn.cluster import KMeans

X, y = make_circles(n_samples=1000, shuffle=True, noise=0.05, random_state=0)
clusterDF = pd.DataFrame(data=X, columns=['ftr1', 'ftr2'])
clusterDF['target'] = y
clusterDF['kmeans_cluster'] = kmeans_labels

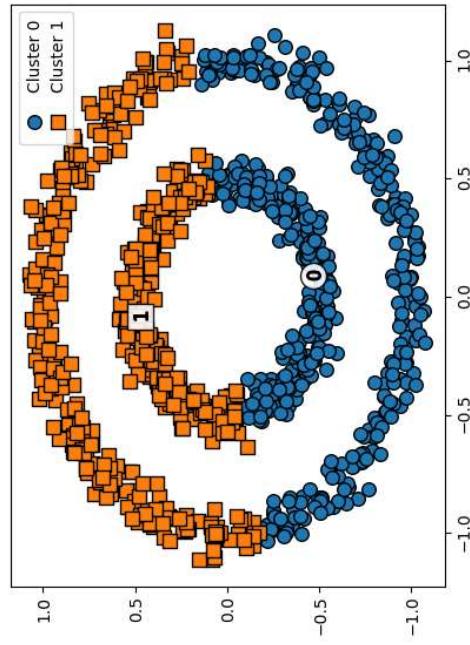
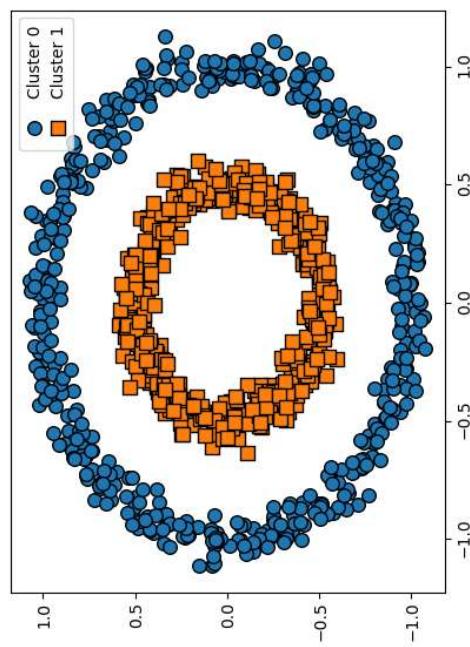
visualize_cluster_plot(kmeans, clusterDF, 'kmeans_cluster', iscenter=True)
```

## (2) K-평균

```
# KMeans은 make_circles() 데이터 세트를 클러스터링 수령.
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=2, max_iter=1000, random_state=0)
kmeans_labels = kmeans.fit_predict(X)
clusterDF['kmeans_cluster'] = kmeans_labels

visualize_cluster_plot(kmeans, clusterDF, 'kmeans_cluster', iscenter=True)
```



# 7.3 DBSCAN 적용하기(make\_circles())

## (3) GMM

```
# GMM으로 make_circles() 데이터셋을 클래스화할 수 있음.
from sklearn.mixture import GaussianMixture

gmm = GaussianMixture(n_components=2, random_state=0)
gmm_label = gmm.fit(X).predict(X)
clusterDF['gmm_cluster'] = gmm_label

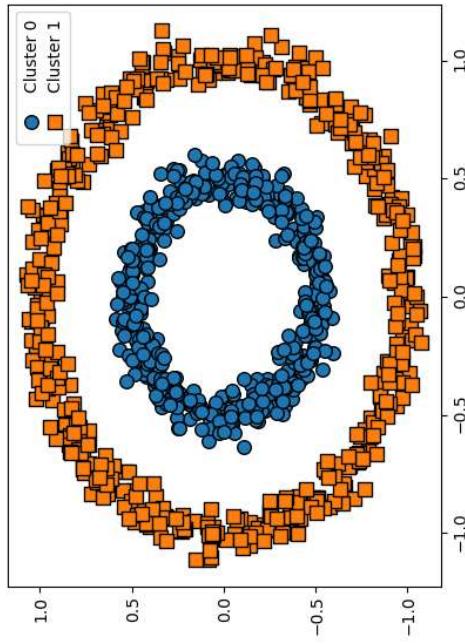
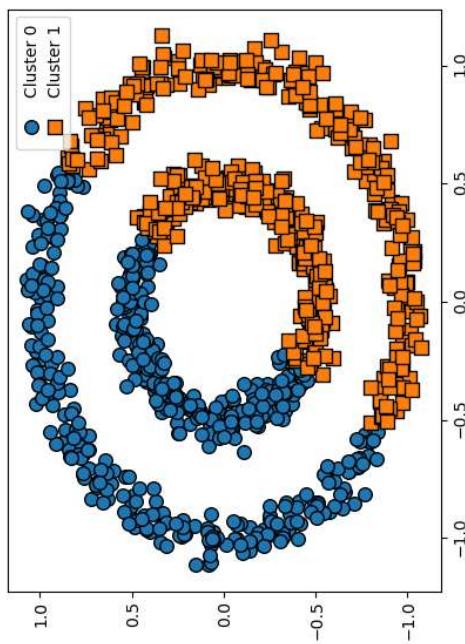
visualize_cluster_plot(gmm, clusterDF, 'gmm-cluster', iscenter=False)
```

## (4) DBSCAN

```
# DBSCAN은 make_circles() 데이터셋을 클러스터링할 수 있음.
from sklearn.cluster import DBSCAN

dbscan = DBSCAN(eps=0.2, min_samples=10, metric='euclidean')
dbscan_labels = dbscan.fit_predict(X)
clusterDF['dbscan-cluster'] = dbscan_labels

visualize_cluster_plot(dbscan, clusterDF, 'dbscan-cluster', iscenter=False)
```





## 8. 뷔이즈 가우시안 혼합 모델

# 8.1 베이즈 가우시안 혼합 모델

최적의 클러스터 개수를 수동으로 찾지 않고 불필요한 클러스터의 가중치를 0으로 만드는 Bayesian Gaussian Mixture 클래스를 사용할 수 있음.

- 클러스터 개수  $n_{components}$ 를 최적의 클러스터 개수보다 크다고 믿을 만한 값으로 지정
- 자동으로 불필요한 클러스터를 제거

# 8.1 베이지안 가우시안 혼합 모델

## (1) 데이터 생성

```

1 import pandas as pd
2 import numpy as np
3 from sklearn.datasets import make_blobs
4 from matplotlib import pyplot as plt
5 X, y = make_blobs(n_samples=350, centers=4, cluster_std=0.60)
6 plt.scatter(X[:, 0], X[:, 1], c='viridis')
7 
```

## (2) 베이지안 가우스 혼합 모델

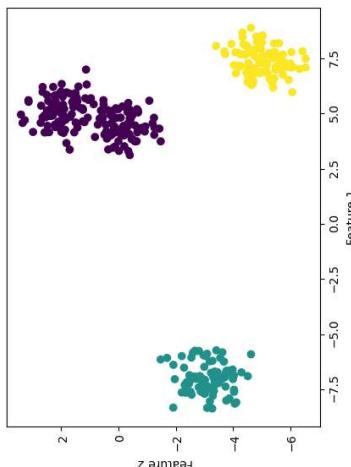
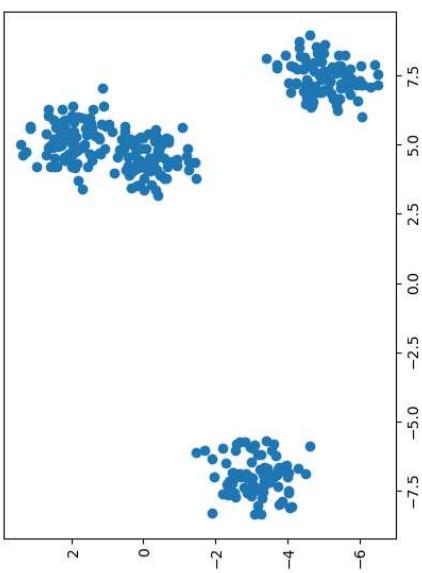
```

1 from sklearn.mixture import BayesianGaussianMixture
2 bgm = BayesianGaussianMixture(n_components = 10, n_init = 10, random_state=42)
3 bgm.fit(X)
4 np.round(bgm.weights_, 2)
5 array([0.5 , 0.25, 0.25, 0. , 0. , 0. , 0. , 0. , 0. ])
6 n_clusters_ = (np.round(bgm.weights_, 2) > 0).sum()
7 n_clusters_
3 
```

## (3) 예측

```

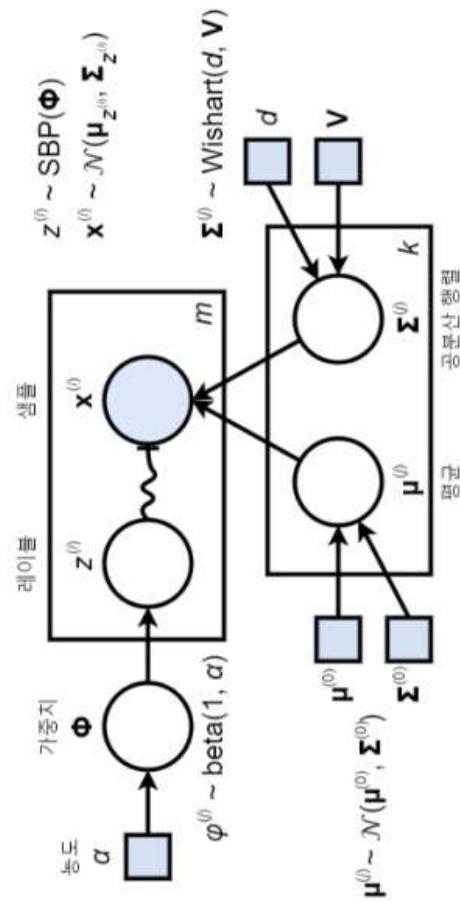
1 y_pred = bgm.predict(X)
2 plt.scatter(X[:, 0], X[:, 1], c=y_pred, cmap="viridis")
3 plt.xlabel("Feature 1")
4 plt.ylabel("Feature 2") 
```



## 8.2 베이즈 가우시안 혼합 모델의 구조

- 클러스터 파라미터(가중치, 평균, 공분산 행렬)는 고정된 파라미터가 아닌 클러스터 할당처럼 잠재 확률 변수로 취급됩니다. 따라서  $z$ 는 클러스터 파라미터와 클러스터 할당을 모두 포함함.

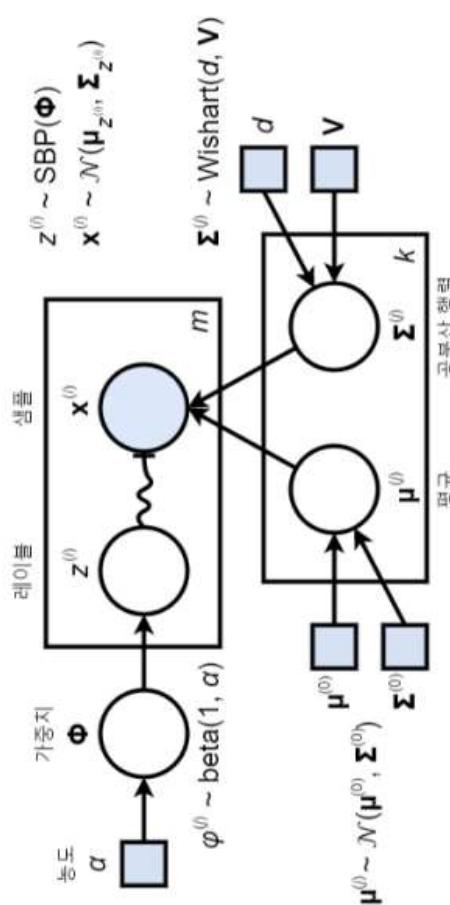
[베이즈 가우시안 혼합 모델의 구조]



- **베타 분포**: 두 매개변수  $\alpha, \beta$ 에 대해  $[0, 1]$ 에서 정의되는 연속확률분포. 고정 범위 안에 놓인 값을 가진 확률 변수를 모델링할 때 사용함.
- **SBP**: 매개변수  $\Phi = [\phi^{(1)}, \dots, \phi^{(m)}]$ 에 대해 샘플의  $\phi^{(1)}$ 을 클러스터 0에, 남은 샘플의  $\phi^{(2)}$  가 클러스터 1에 할당하는 식으로 이루어지는 분포. 베타 분포의 다변수로의 확장을 디리클레 분포라고 하며 SBP는 디리클레 분포를 무한대로 확장하는 디리클레 과정에서 샘플 본을 추출하기 위한 방법론으로 이용됨. 새로운 운율 표본이 작은 샘플이 높은 네이터셋에 잘 맞음.

## 8.2 베이즈 가우시안 혼합 모델의 구조

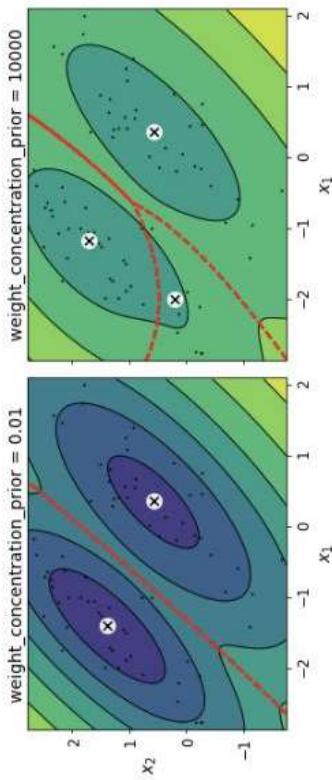
[베이즈 가우시안 혼합 모델의 구조]



- 놓도 :  $\alpha$ 가 크면,  $\Phi$  값이 0에 가까워지며 SBP는 이에 따라 많은 클러스터를 만들어 냄. 반대로  $\alpha$ 가 작아지면  $\Phi$  값이 1에 가까워지며 몇 개의 클러스터만 만들어짐.
- 놓도 :  $\alpha$ 가 크면,  $\Phi$  값이 0에 가까워지며 SBP는 이에 따라 많은 클러스터를 만들어 냄. 반대로  $\alpha$ 가 작아지면  $\Phi$  값이 1에 가까워지며 몇 개의 클러스터만 만들어짐.
- 위사트 분포 : 다변량 정규분포를 따르는  $n$ 개의 확률 벡터  $X_1, X_2, \dots, X_n \sim_{\text{iid}} \mathcal{N}(0, V)$ 에 대해  $\sum_{i=1}^n \gamma_i X_i X_i^T \sim \text{Wishart}(n, V)$ 를 이용. 이를 사용해 공분산 행렬을 샘플링하며 파라미터  $d$ 와  $V$ 가 클러스터의 분포 모양을 제어함.

# 8.3 클러스터 개수 조정

[다른 농도 값으로 동일한 데이터에서 다른 개수의 클러스터]



잠재 변수  $z$ 에 대한 사전 확률  $p(z)$ 에 사전 믿음을 인코딩하여 클러스터 개수를 조정할 수 있음.  
 $\text{Weight\_concentration\_prior}$  매개변수를 크게 설정하면 클러스터 개수가 풍부하다고 믿는 것(높은 농도)이며 낮게 설정하면 클러스터가 적을 것이라고 믿는 것(낮은 농도, 대부분의 가중치를 일부 클러스터에 부여)임.

# 8.4 베이즈 정리

관측된 데이터  $\mathbf{x}$ 로 잠재 변수에 대한 확률을 분포 갱신 : 베이즈 정리를 사용하여 사후 확률(posterior,  $X$ 가 주어졌을 때  $z$ 의 조건부 확률) 분포를 계산

$$p(\mathbf{z} \mid \mathbf{X}) = \text{사후 확률} = \frac{\text{가능도} \times \text{사전 확률}}{\text{증거}} = \frac{p(\mathbf{X} \mid \mathbf{z})p(\mathbf{z})}{p(\mathbf{X})}$$

베이즈 통계학의 주요 문제 : 가우시안 혼합과 같은 많은 경우에 분모인  $P(X)$ 를 구하려면 가능한 모든  $Z$ 값에 대해(모든 클러스터 파라미터와 클러스터 할당의 조합을 고려하여) 적분해야 하므로 실질적으로 계산이 어려움.

$$p(\mathbf{X}) = \int p(\mathbf{X} \mid \mathbf{z})p(\mathbf{z})d\mathbf{z}$$

해결 방법 : 변분 추론 사용.



## 9. 클러스터 개수 선택하기

# 9.1 BIC, AIC

가우시안 혼합에서는 클러스터가 타원형이며 크기가 다르므로 이너셔나 실루엣 점수를 사용하기 힘듦.  
이론적 정보 기준 : 모델 선택을 위한 통계 모델의 측정치. 복잡한 모델(클러스터가 많은)에게는 별점을,  
데이터에 잘 학습하는 모델에 보상을 주는 방식으로 측정됨. 가우시안 혼합에서는 정의된 이론적 정보  
기준을 최소화하는 모델을 찾으며 대표적으로 BIC나 AIC를 사용함.

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

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

- BIC, AIC는 종종 동일한 모델을 선택
- 둘의 선택이 다를 경우 BIC 모델이 AIC가 선택한 모델보다 간단한(=파라미터가 적은) 경향이 있음.
- (단점) (특히 대규모 데이터셋에서) 데이터가 아주 잘 맞지 않을 수 있음.

# 9.1 BIC, AIC

## (1) 데이터 생성

```

1 x1, y1 = make_blobs(n_samples=1000, centers=((4, -4), (0, 0)), random_state=42)
2 x1 = x1.dot(np.array([[0.374, 0.95], [0.732, 0.598]]))
3 x2, y2 = make_blobs(n_samples=250, centers=[], random_state=42)
4 x2 = x2 + [6, -8]
5 x = np.r_[x1, x2]
6 y = np.r_[y1, y2]

```

## (2) GMM

```

1 from sklearn.mixture import GaussianMixture
2 gm = GaussianMixture(n_components=3, n_init=10, random_state=42)
3 gm.fit(X)

```

## (3) BIC / AIC

```

1 print(gm.bic(X))
2 print(gm.aic(X))

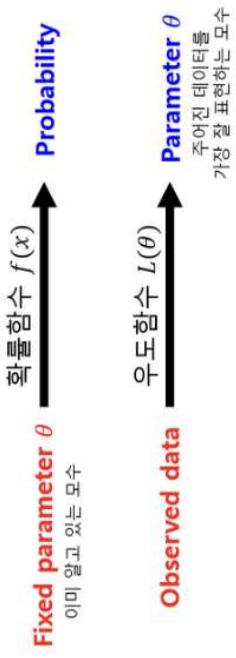
```

8189.747000497187  
8102.521720382149

# 9.2 가능도 합수

가능도(우도) : 특정한 현상이 일어났다고 할 때, 특정 통계 모델 하에서 이 현상이 일어날 확률  
 확률과 가능도는 종종 구별 없이 사용되지만, 통계학에서는 이 둘은 다른 의미를 가짐.  
 (확률 합수가 되려면 모든 확률의 합은 항상 1이 되어야 하지만, 우도 함수 값은 더하면, 1이 되지 않음)

- <ex>
- 확률은 출력  $x$ 의 합수로 파라미터  $\theta$ 를 알고 있을 때 미래의 출력  $x$ 가 얼마나 그럴듯한지 설명함.
  - 가능도는 파라미터  $\theta$ 의 합수로 출력  $x$ 를 알고 있을 때 특정 파라미터 값  $\theta$ 가 얼마나 그럴듯한지 설명함.



# 9.2 가능성도 함수

최대 가능 추정(MLE) : 주어진 데이터셋에 대하여 가능도 함수를 최대화하는 파라미터  $\theta$  찾기.

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta | \mathbf{x})$$

최대 사후 확률(MAP) : 파라미터에 대한 사전 확률 분포  $g$ 가 존재할 경우  $\mathcal{L}(\theta | \mathbf{x}) g(\theta)$ 를 최대화하는 파라미터  $\theta$  찾기. MAP가 파라미터 값을 제약하므로 MLE의 규제 버전으로 생각할 수 있음.

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}} \mathcal{L}(\theta | \mathbf{x}) g(\theta)$$

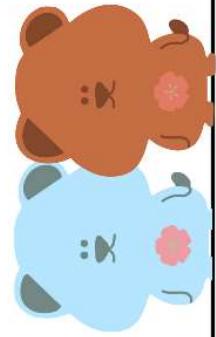
가능도 함수 최대화 : 가능도 함수의 최대화는 이 함수의 로그를 최대화하는 것과 동일하며, 일반적으로 로그 가능도를 최대화하는 것이 더 쉬우므로 가능도를 로그로 바꾸어 계산함. 최대값  $\hat{\theta}$ 를 추정하면, AIC와 BIC를 계산하기 위해 필요한 값  $\mathcal{L}(\hat{\theta} | \mathbf{x})$ 를 계산할 수 있으며, 이를 모델이 데이터에 얼마나 잘 맞는지 측정하는 값으로 생각할 수 있음.

출처 : 우도 함수(likelihood function)의 이해 (tistory.com).

출처 : [해즈로 머신러닝] Ch. 9 비지도 학습 9. : 네이버 블로그 (naver.com)

출처 : [해즈로 머신러닝] 100. 우도 가능도(학습)... : 네이버 블로그 (naver.com)

출처 : [해즈로 머신러닝] 9장 - 비지도 학습 2. 간우시안 혼합 (tistory.com)



THANK YOU