# 군집 (clustering)

특성만 알고 있는 상황에서 훈련과 모델 평가에 필요한 타깃 데이터가 없기 때문에 지도 학습을 사용할 수 없는 경우 비지도 학습을 사용합니다.
군집(clustering) 알고리즘의 목적은 샘플에 잠재되어 있는 그룹을 식별하는 것입니다.

## ➢ K-Means 클러스터링

- 레이블이 없는 데이터 안에서 패턴과 구조를 발견하는 비지도 학습
- 중심 기반 클러스터링 알고리즘 - "동일한 클래스에 속하는 데이터는 어떠한 중심을 기준으로 분포할 것이다"

- 클러스터링, 군집화를 사용하는 예 :
  추천 엔진 : 사용자 경험을 개인화하기 위해 비슷한 제품 묶어주기
  검색 엔진: 관련 주제나 검색 결과 묶어주기
  시장 세분화(segmentation): 지역, 인구 통계, 행동에 따라 비슷한 고객들 묶어주기

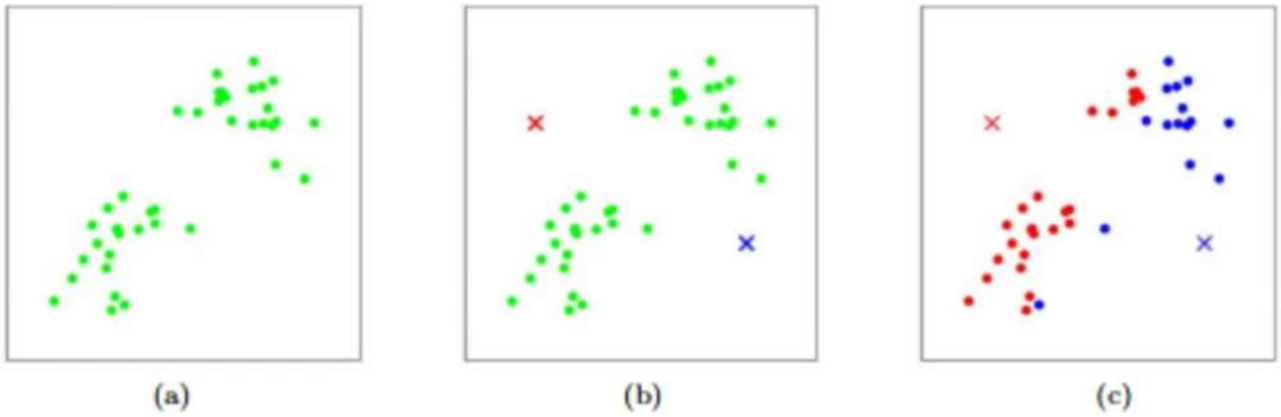
- 군집화의 목표
  서로 유사한 데이터들은 같은 그룹으로, 서로 유사하지 않은 데이터는 다른 그룹으로 분리하는 것

- K-Means 알고리즘 :
  몇개의 그룹으로 묶을 것인가
  데이터의 "유사도"를 어떻게 정의할 것인가 (유사한 데이터란 무엇인가)
  해결할 수 있는 가장 유명한 전략

## ➢ K-Means 클러스터링

- "K"는 데이터 세트에서 찾을 것으로 예상되는 클러스터(그룹) 수를 말한다.
- "Means"는 각 데이터로부터 그 데이터가 속한 클러스터의 중심까지의 평균 거리를 의미한다. (이 값을 최소화하는 게 알고리즘의 목표가 된다.)

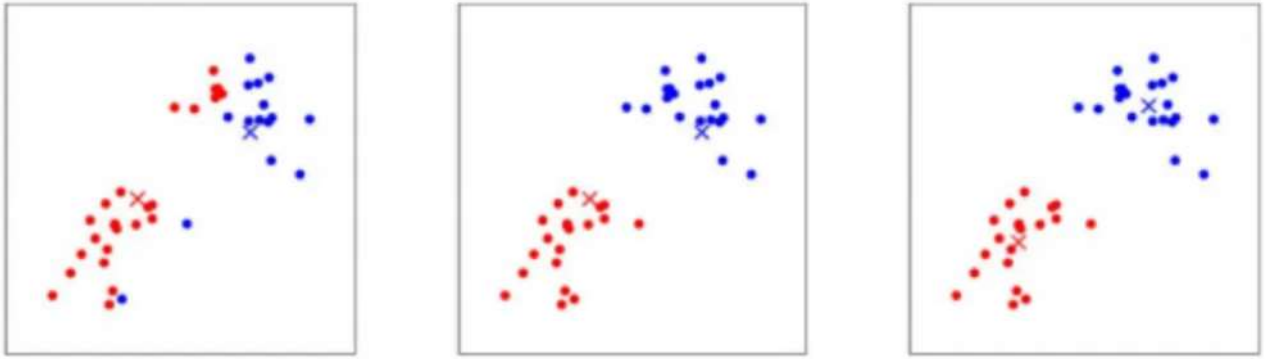
1. 일단 K개의 임의의 중심점(centroid)을 배치하고
2. 각 데이터들을 가장 가까운 중심점으로 할당한다. (일종의 군집을 형성한다.)
3. 군집으로 지정된 데이터들을 기반으로 해당 군집의 중심점을 업데이트한다.
4. 2번, 3번 단계를 그래서 수렴이 될 때까지, 즉 더이상 중심점이 업데이트 되지 않을 때까지 반복한다.




(b)에서 일단 중심점 2개를 아무 데나 찍고, (c)에서는 각 데이터들을 두 개 점 중 가까운 곳으로 할당한다.

## ➢ K-Means 클러스터링

•

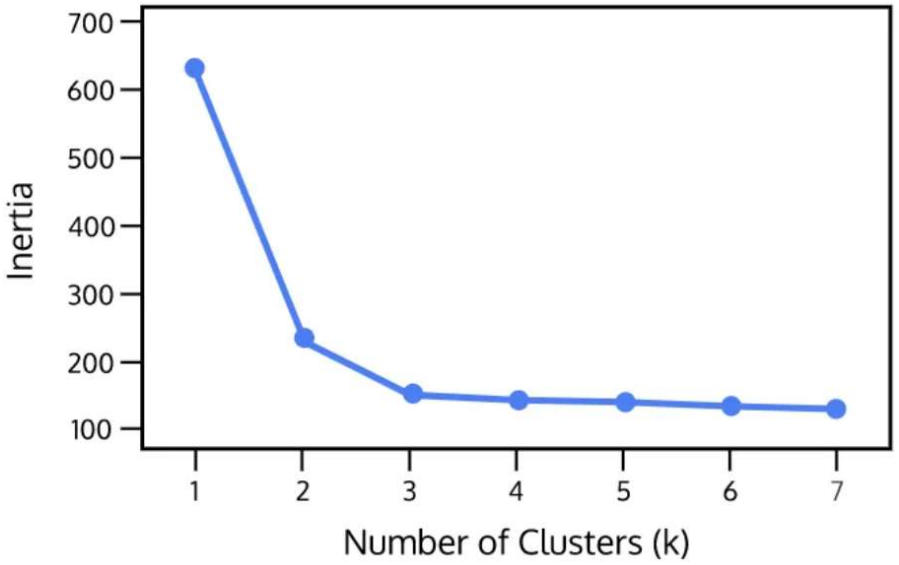



(d)에서는 그렇게 군집이 지정된 상태로 중심점을 업데이트 한다.
(e)에서는 업데이트 된 중심점과 각 데이터들의 거리를 구해서 군집을 다시 할당한다.

군집화를 해놓으면 새로운 데이터가 들어와도 그게 어떤 군집에 속할지 할당해줄 수 있게 된다

## ➢ K-Means 클러스터링

- 군집화를 하기 위해서는 몇개의 군집이 적절할지 결정해야 하는데, 그러려면 일단 "좋은 군집"이란 무엇인지 정의할 수 있어야 한다.
- 군집화가 잘 되었면 각 군집의 샘플이 가까운 거리에서 오밀조밀하게 묶인다.
- 데이터들이이 얼마나 퍼져 있는지 (혹은 얼마나 뭉쳐있는지) 응집도는 inertia 값으로 확인한다.
- inertia는 각 데이터로부터 자신이 속한 군집의 중심까지의 거리를 의미하기 때문에 inertia 값이 낮을수록 군집화가 더 잘 됐다고 볼 수 있다.
- 군집의 개수, 즉 k 값을 바꿔가면서 inertia를 그래프로 표시하면 다음 모양의 그래프가 된다.




k값이 증가하면 intertia 값은 감소하다가 어느정도 수준이 되면 거의 변화를 안 보이게 된다.
대부분의 경우 너무 많지 않은 군집으로 분류하면서도 intertia 값이 작은 상태가 최선이 될 거다.
그래프의 예에서는 최적의 클러스터 수는 3으로 보인다.

➢ **K-Means++ 클러스터링**

- K-Means 알고리즘의 가장 큰 단점은 처음에 지정하는 중심점(centroid)의 위치를 무작위로 결정하기 때문에 최적의 클러스터로 묶어주는 데에는 한계가 있다
- K-Means++는 전통적인 K-Means의 문제, 즉 중심점 무작위 선정의 문제를 해결하기 위한 알고리즘입니다.
- K-Means에서 가장 첫번째 단계, 즉 중심점을 배치하는 걸 그냥 임의로 하는 대신 좀 더 신중하게 배치하는 거다.

1. 가지고 있는 데이터 포인트 중에서 무작위로 1개를 선택하여 그 녀석을 첫번째 중심점으로 지정한다.
2. 나머지 데이터 포인트들에 대해 그 첫번째 중심점까지의 거리를 계산한다.
3. 두번째 중심점은 각 점들로부터 거리비례 확률에 따라 선택한다. (즉, 이미 지정된 중심점으로부터 최대한 먼 곳에 배치된 데이터포인트를 그 다음 중심점으로 지정한다는 뜻이다.)
4. 중심점이 k개가 될 때까지 2, 3번을 반복한다.

- k-means로 중심점을 랜덤하게 지정
- K-means(n_clusters=k,init='k-means++' )
- 초기 중심점을 더욱 전략적으로 배치하기 때문에 전통적인 K-Means보다 더 최적의 군집화를 할 수 있다.
- K-Means보다 알고리즘이 수렴하는 속도가 빠르다.

# 군집 (clustering)

- K-평균을 사용한 군집
  - 샘플을 k 개의 그룹으로 나눌때 k-평균(k-means) 군집을 사용합니다.
  - k-평균(k-means) 군집 알고리즘은 샘플을 k개의 그룹으로 나눕니다. 각 그룹은 거의 동일한 분산을 가집니다.
  - 그룹의 개수 k는 하이퍼파라미터로 사용자가 지정해야 합니다.

  - k-평균 동작 방식

1. k개의 클러스터 '중심' 포인트를 랜덤한 위치에 만듭니다.
2. 각 샘플에 대해
   a. 각 샘플과 k개의 중심 포인트 사이 거리를 계산합니다.
   b. 샘플을 가장 가까운 중심 포인트의 클러스터에 할당합니다.
3. 중심 포인트를 해당하는 클러스터의 평균(중심)으로 이동합니다.
4. 더 이상 샘플의 클러스터 소속이 바뀌지 않을 때까지 단계 2와 단계 3을 반복합니다

  - k-평균 군집은 클러스터가 둥그런 모양으로 간주합니다.
  - k-평균 군집은 모든 특성은 동일한 스케일을 가정합니다. (가정에 맞도록 특성을 표준화해야 합니다.)
  - k-평균 군집은 클러스터 크기는 균형 잡혀 있습니다.

# 군집 (clustering)

- K-평균을 사용한 군집
  - 사이킷런에는 KMeans 클래스에 k-평균 군집이 구현되어 있습니다
  - n_clusters 매개변수가 클러스터 k의 수를 지정합니다.
  - labels_ 속성에서 각 샘플의 예측 클래스를 확인할 수 있습니다.

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import KMeans

iris = datasets.load_iris()    # 데이터  로드
features = iris.data

scaler = StandardScaler()  # 특성 표준화
features_std = scaler.fit_transform(features)

cluster = KMeans(n_clusters=3, random_state=0, n_jobs=-1)  # k-평균 객체 생성
model = cluster.fit(features_std)    # 모델  훈련
model.labels_           # 예측 클래스 확인
iris.target             # 진짜 클래스  확인

new_observation = [[0.8, 0.8, 0.8, 0.8]]   #New Sample Data
model.predict(new_observation)    # 샘플의 클러스터를 예측
model.cluster_centers_
```

# 군집 (clustering)

- K-평균을 사용한 군집
  - K-Means++ 알고리즘은 중심 포인트 하나를 먼저 랜덤하게 선택하고 그 다음부터는 이전 중심 포인트와의 거리를 고려하여 다음 중심 포인트를 선택합ㄴ디ㅏ.
  - 사이킷런의 KMeans 클래스의 init 매개변수 기본값은 k-means++입니다
  - n_init 횟수만큼 반복하여 최상의 결과를 만드는 중심 포인트를 찾습니다. (기본값 10)
  - 비교하는 기준은 샘플과 클러스터 중심까지의 거리 합(이너셔, inertia_속성에 저장)입니다. (score 메서드에서 반환하는 값)
  - 샘플 데이터를 각 클러스터까지 거리로 변환하는 transform()를 제공합니다.

```
model.inertia_
model.score(features_std)
model.transform(new_observation)

inertia = []
for n in range(1, 10):
    kmeans = KMeans(n_clusters=n, random_state=0, n_jobs=-1)
    inertia.append(kmeans.fit(features_std).inertia_)

import matplotlib.pyplot as plt
plt.plot(range(1, 10), inertia)
plt.show()
```

# 군집 (clustering)

- K-평균 군집 속도 향상
  - 미니배치 k-평균은 랜덤 샘플에 대해서만 수행합니다. (성능을 조금만 희생하고 알고리즘 학습에 드는 시간을 대폭 줄여줍니다.)
  - batch_size 매개변수는 각 배치에 랜덤하게 선택할 샘플의 수를 조절합니다
  - 훈련 세트가 너무 크면 하나의 넘파일 배열로 전달하기 어려우며, 데이터를 조금씩 전달하면서 훈련할 수 있는 partial_fit()를 사용합니다.
  - 실전에서는 파일 같은 외부 저장소에서 시스템의 메모리가 허용하는만큼 데이터를 추출하여 모델을 훈련합니다

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import MiniBatchKMeans

iris = datasets.load_iris()  # 데이터 로드
features = iris.data
scaler = StandardScaler()  # 특성을 표준화
features_std = scaler.fit_transform(features)

cluster = MiniBatchKMeans(n_clusters=3, random_state=0, batch_size=100)
model = cluster.fit(features_std)  # 모델 훈련

mb_kmeans = MiniBatchKMeans()
for i in range(3):
    mb_kmeans.partial_fit(features_std[i*50:(i+1)*50])
```

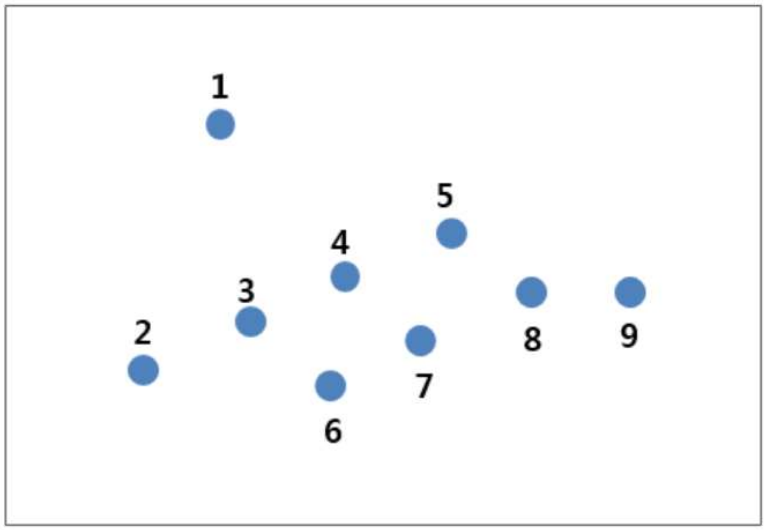
# 군집 (clustering)

- 평균이동을 사용한 군집
  - 사이킷런의 평균 이동 구현인 MeanShift는 클러스터 수나 모양을 가정하지 않고 샘플을 그룹으로 나눌때 사용합니다
  - bandwidth는 샘플이 이동 방향을 결정하기 위해 사용하는 면적(커널)의 반경을 지정합니다.

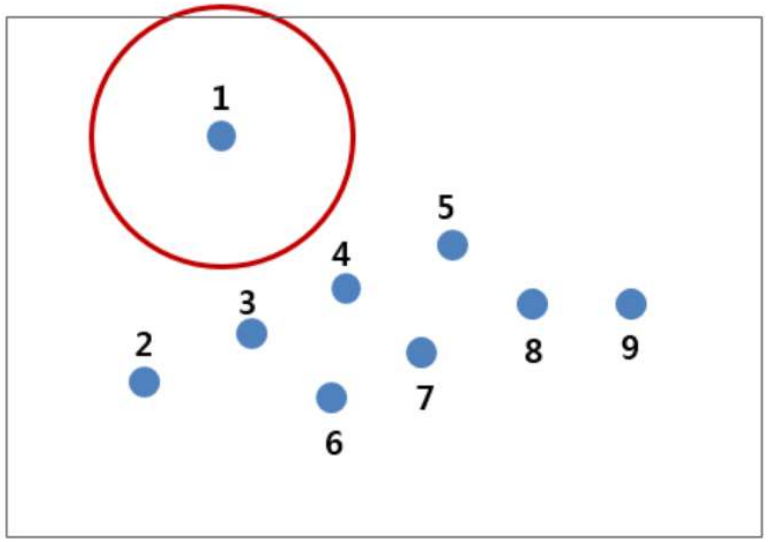
```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import MeanShift

iris = datasets.load_iris()  # 데이터 로드
features = iris.data

scaler = StandardScaler()  # 특성을 표준화
features_std = scaler.fit_transform(features)
cluster = MeanShift(n_jobs=-1)  # meanshift 객체 생성
model = cluster.fit(features_std)  # 모델 훈련

model.labels_
model.cluster_centers_
```

➢ **DBSCAN(Density-based Spatial Clustering of Applications with Noise)**

- DBSCAN은 각각의 데이터들에 대해 이웃한 데이터와의 밀도를 계산하면서 불특정한 모양의 클러스터를 생성한다.



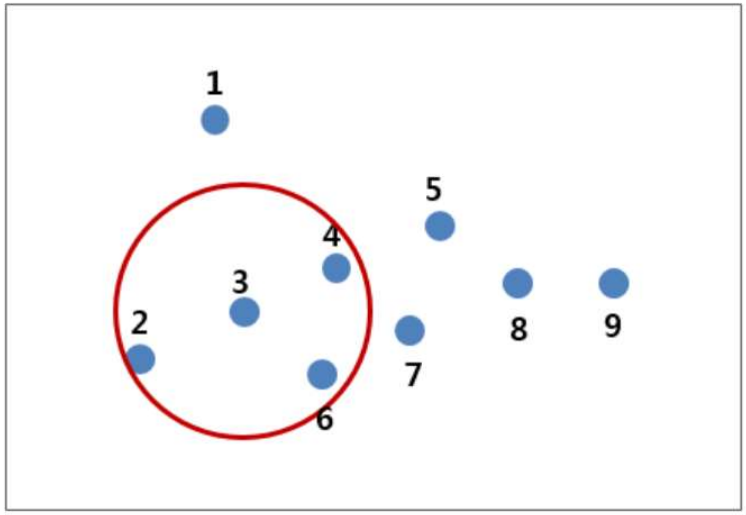

반지름이 e인 원이 있다고 하고, 1번 데이터로부터 9번 데이터까지 원의 중심에 이 데이터들을 둔다



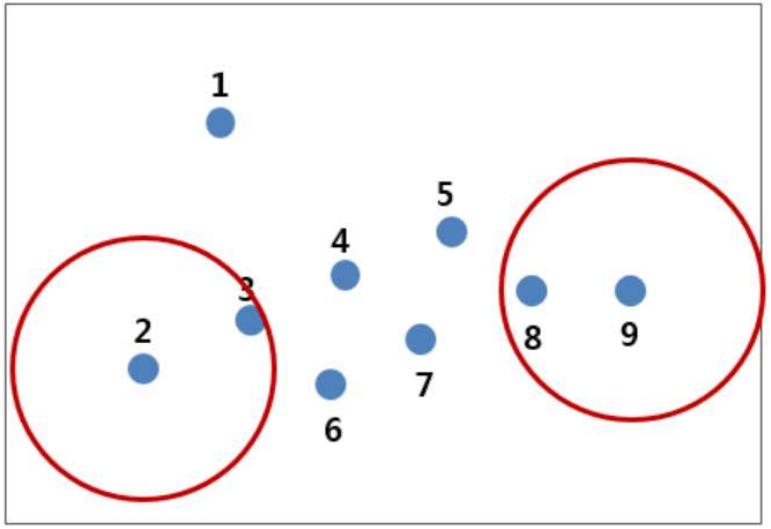

원안에 적어도 4개 이상의 점들이 포함되는 점(데이터)를 원에 중심에 두었을 경우 1번 데이터를 제외하고는 아무런 데이터가 없습니다.
이런 데이터들을 노이즈 데이터(noise point)라 부르며, 노이즈 데이터들은 클러스터에서 제외시킵니다.

➢ **DBSCAN(Density-based Spatial Clustering of Applications with Noise)**

- 각각의 데이터들에 대해 이웃한 데이터와의 밀도를 계산하면서 불특정한 모양의 클러스터를 생성한다.
- DBSCAN을 정의하기 위해서는 이웃한 데이터와 클러스터에 대한 정의가 필요하다.
- 이를 위해 DBSCAN의 hyper-parameter로 주어지는 주변 거리 $\epsilon$과 최소 이웃 데이터의 수 nc를 기반



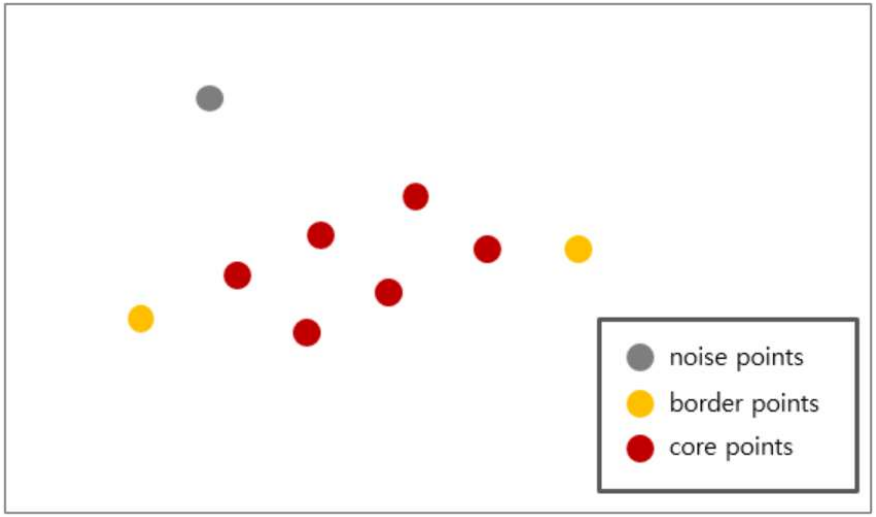

3번 데이터를 원의 중심으로 두었을 때 4개의 점이 원안에 포함됩니다.
기준에 충족하는 데이터가 되며 이러한 데이터를 코어 데이터(core point)라 부릅니다.
코어 데이터들은 하나의 클러스터에 포함시킵니다.




2번과 9번 데이터는 기준인 4개의 데이터를 포함하는 것에는 충족하지 못하지만 클러스터에 포함되는 코어 데이터인 3번 데이터를 포함하고 있습니다.
이런 데이터를 경계 데이터(border point)라 부르며 해당 클러스터에 포함시킵니다.

## ➢ DBSCAN(Density-based Spatial Clustering of Applications with Noise)

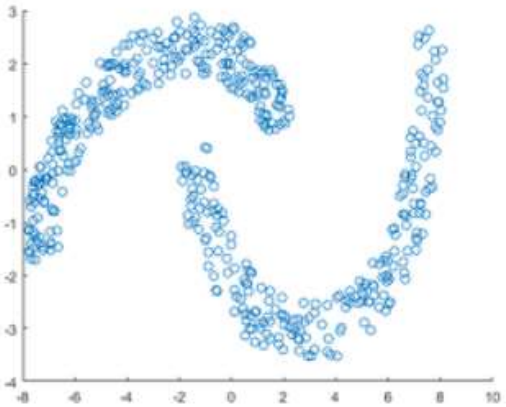
- DBSCAN은 위에서 정의된 개념을 바탕으로 주어진 데이터에 대해 밀도를 기반으로 개수와 형태가 정해지지 않은 클러스터들을 형성하고, 노이즈 데이터를 분류하는 알고리즘이다



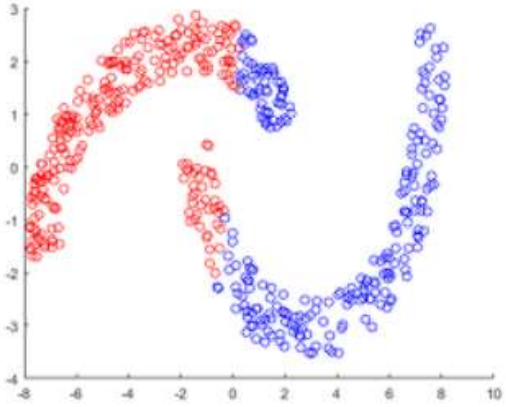

반지름 e인 원을 이용해 코어 데이터, 경계 데이터, 노이즈 데이터들을 분류한 결과
빨간색과 노란색 데이터들을 하나의 클러스터로 묶고, 회색 데이터는 클러스터에서 제외합니다.

➢ **DBSCAN(Density-based Spatial Clustering of Applications with Noise)**

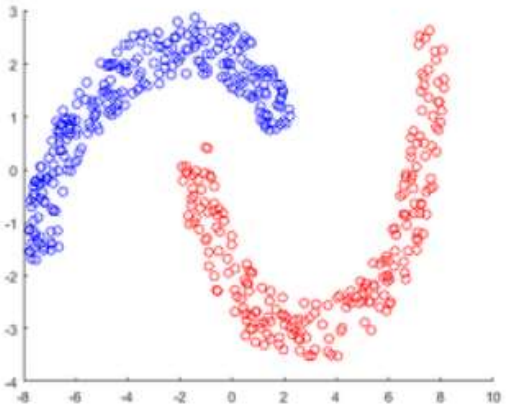
- 중심 기반 클러스터링 알고리즘은 클러스터의 중심을 기준으로 가까운 데이터들을 클러스터에 포함시키기 때문에 원의 형태로 클러스터의 모양이 형성된다.
- 반면에 밀도 기반 클러스터링 알고리즘은 서로 이웃한 데이터들을 같은 클러스터에 포함시키기 때문에 불특정한 모양의 클러스터가 형성된다.
- Gaussian 분포가 아닌, 불특정한 분포를 따르는 데이터에 대해서는 밀도 기반 클러스터링 알고리즘을 활용하는 것이 적절하다.



(a) 원본 데이터



(b) k-means clustering의 결과



(c) DBSCAN의 결과

➢ **DBSCAN(Density-based Spatial Clustering of Applications with Noise)**

- k-means clustering과 다르게 클러스터의 수를 지정할 필요가 없으며, 알고리즘이 자동으로 클러스터의 수를 찾는다.
- 원 모양의 클러스터뿐만 아니라, 불특정한 모양의 클러스터도 찾을 수 있다.
- 클러스터링을 수행하는 동시에 노이즈 데이터도 분류할 수 있기 때문에 outlier에 의해 클러스터링 성능이 하락하는 현상을 완화할 수 있다.
- k-means clustering은 데이터의 수에 대해 linear time complexity 갖지만, DBSCAN은 quadratic time complexity를 갖는다.
- 알고리즘이 이용하는 거리 측정 방법에 따라 클러스터링 결과가 변한다.
- 데이터의 특성을 모를 경우에는 알고리즘의 적절한 hyper-parameter를 설정하기가 어렵다.

# 군집 (clustering)

- DBSCAN을 사용한 군집
  - DBSCAN은 샘플의 밀집 영역을 클러스터로 그룹핑할 수 있습니다.
  - eps매개변수 : 다른 샘플을 이웃으로 고려하기 위한 최대 거리
  - min_samples : 핵심 샘플로 간주하기 위해 eps 거리 내에 필요한 최소 샘플 개수
  - metric : eps에서 사용할 거리 측정 방식
  - DBSCAN에서 찾은 핵심 샘플의 인덱스는 core_sample_indices_속성에 저장되어 있습니다.
  - 훈련 데이터에 대한 예측 결과를 얻으려면 fit_predict()를 사용합니다.

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import DBSCAN

iris = datasets.load_iris()  # 데이터 로드
features = iris.data
scaler = StandardScaler()  # 특성 표준화
features_std = scaler.fit_transform(features)

cluster = DBSCAN(n_jobs=-1)      # DBSCAN 객체 생성
model = cluster.fit(features_std)  # 모델 훈련
model.labels_        # 클러스터 소속을 확인
model.core_sample_indices_
cluster.fit_predict(features_std)
```

# 군집 (clustering)

- 계층적 병합을 사용한 군집
  - 병합 군집(agglometrative clustering)은 강력하고 유연한 계층적 군집 알고리즘입니다.
  - 병합 군집은 모든 샘플이 각자 하나의 클러스터로 시작합니다. 그 다음 어떤 조건에 부합하는 클러스터들이 서로 병합됩니다.
  - 이 과정이 어떤 종료 조건에 도달할 때까지 반복되어 클러스터가 커집니다.
  - 사이킷런의 AgglomerativeClustering 클래스는 linkage 매개변수를 사용하여 병합된 클러스터의 분산(ward) 또는 두 클러스터 샘플 간의 평균 거리(average) 또는 두 클러스터 샘플 간의 최대 거리(complete)를 최소화하는 병합 전략을 결정합니다.
  - affinity매개변수는 linkage에서 사용할 거리 측정 방식을 결정합니다. (minkowski, euclidean등)
  - n_clusters는 군집 알고리즘이 찾을 클러스터 수를 지정합니다.
  - n_clusters개의 클러스터가 남을 때까지 연속적으로 병합됩니다
  - labels_속성을 사용해 각 샘플이 속한 클러스터를 확인할 수 있습니다.
  - linkage 매개변수에 두 클러스터 샘플 간의 최소 거리를 최소화하는 병합 전략인 single 옵션이 추가되었습니다.

# 군집 (clustering)

- 계층적 병합을 사용한 군집

```
from sklearn import datasets
from sklearn.preprocessing import StandardScaler
from sklearn.cluster import AgglomerativeClustering

iris = datasets.load_iris()   # 데이터 로드
features = iris.data

scaler = StandardScaler()   # 특성  표준화
features_std = scaler.fit_transform(features)

cluster = AgglomerativeClustering(n_clusters=3)   # 병합 군집 객체 생성
model = cluster.fit(features_std)    # 모델  훈련
model.labels_            # 클러스터 소속을 확인
cluster.fit_predict(features_std)
```