7장 군집화

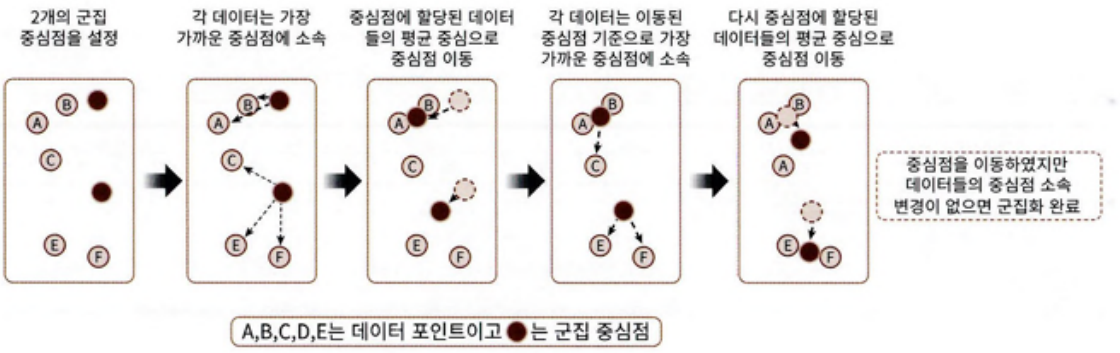
날짜	@2025년 11월 17일
카테고리	개념 정리

01 K-평균 알고리즘 이해

K-평균(K-Means) 개념

K-평균은 가장 널리 쓰이는 **군집화(Clustering)** 알고리즘

1. **임의의 K개의 중심점(centroid)을 초기화**
2. 각 데이터 포인트를 가장 가까운 중심점에 **할당(assign)**
3. 중심점을 해당 군집에 속한 데이터의 **평균 위치로 이동**
4. 중심점 이동 → **다시 데이터 재할당**
5. 더 이상 중심점이 이동하지 않을 때까지 반복



K-평균의 장단점

장점

- 가장 대표적으로 사용되는 군집화 알고리즘
- 개념이 쉽고 구현이 간단
- 대규모 데이터에도 비교적 빠른 계산 가능

단점

- 거리 기반 알고리즘 → **차원 수가 많으면 정확도 저하** → 필요시 PCA 같은 차원 축소 필요
- 초기 중심 설정에 따라 결과가 달라질 수 있음

- 반복 횟수가 많아지면 수행 시간이 증가
- 적절한 **K(군집 개수)를 선택하기 어려움**

사이킷런 KMeans 클래스

주요 파라미터

- **n_clusters** : 군집 개수(K 값)
- **init** : 초기 중심점 설정 방식 (일반적으로 `"k-means++"` 사용)
- **max_iter** : 최대 반복 횟수
- **labels_** : 각 샘플이 속한 군집 번호
- **cluster_centers_** : 군집 중심점 좌표

K-평균을 이용한 붓꽃 데이터 세트 군집화 (예제)

군집화 알고리즘 테스트를 위한 데이터 생성 (예제)

API

여러 개의 클래스에 해당하는 데이터 세트를 만듦

- `make_blobs()` : 개별 군집의 중심점과 표준 편차 제어 기능

  파라미터

  - `n_samples` 생성할 총 데이터 개수 (디폴트 100)
  - `n_feature` 데이터 피처 개수 (시각화 목표시 2)
  - `centers` int 정수값일 때는 군집의 개수 / ndarray형태시 개별 군집 중심점의 좌표
  - `cluster_std` 생성될 군집 데이터의 표준편차
- `make_classification` : 노이즈를 포함한 데이터 만들기

---

02 군집 평가 Cluster Evaluation

군집 평가가 어려운 이유

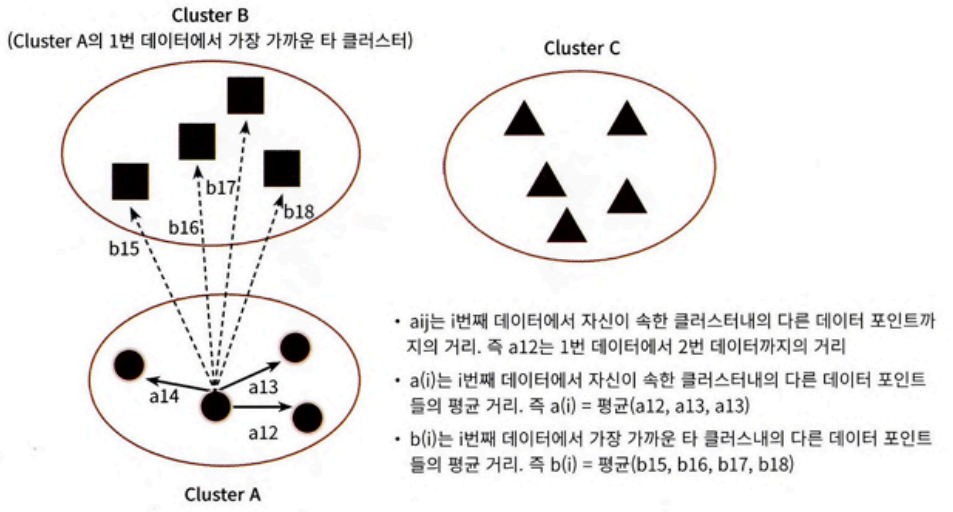
- 군집화는 비지도학습이므로 **정답 레이블(타깃)이 없는 경우**가 대부분
- 분류와 달리, 군집화는 데이터 내부의 숨은 구조를 찾는 것이므로

  정확한 성능 평가 기준을 정의하기 어려움

- 그럼에도 군집의 품질을 평가하기 위해 **실루엣 분석(Silhouette Analysis)**이 널리 사용된다

실루엣 분석

= 실루엣 계수는 *각 데이터가 군집 내에서는 얼마나 가깝고, 다른 군집과는 얼마나 멀리 떨어졌는지*



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

특정 데이터 i에 대해,

- **a(i)** : 같은 군집 내 다른 데이터들과의 평균 거리
- **b(i)** : i가 속하지 않은 군집 중 가장 가까운 군집까지의 평균 거리

값

- **1에 가까움** → 군집 분리가 매우 잘됨
- **0 근처** → 다른 군집과 경계가 모호함
- **음수** → 잘못된 군집에 속해 있을 가능성 있음

좋은 군집의 조건

1. 전체 평균 실루엣 계수는 **0~1 사이**, 1에 가까울수록 좋음
2. **각 군집의 실루엣 평균값 편차가 작아야 함**
    - 특정 군집만 높고 나머지가 낮다면 좋은 군집화 아님

붓꽃 데이터 세트를 이용한 군집 평가 - 실루엣 분석으로

- **silhouette_samples(X, labels)**

  → 각 데이터의 실루엣 계수 반환

- **silhouette_score(X, labels)**

  → 전체 실루엣 계수 평균값 반환 (군집화 품질의 전반적 지표)

군집별 평균 실루엣 계수와 시각화를 통한 군집 개수 최적화 방법

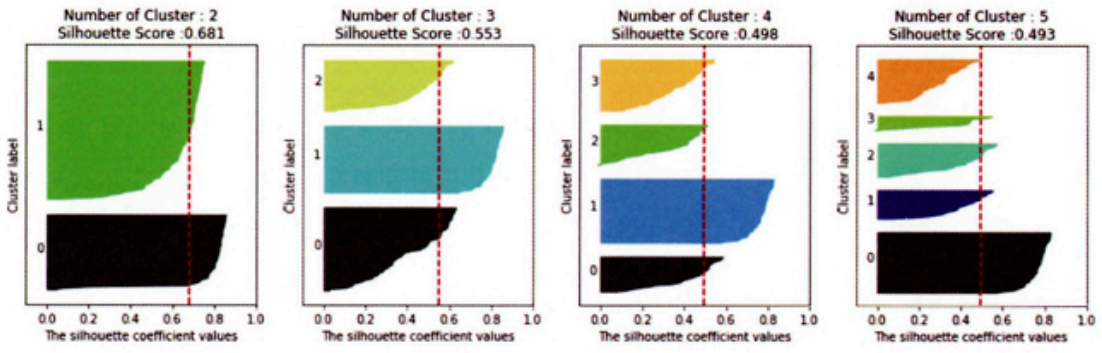
군집 수(k)를 바꾸면서 실루엣 분석을 하면 **최적의 k**를 찾을 수 있다

예시 해석

- **k = 2**
  - 평균 실루엣 값 매우 높음 (0.704)
  - 하지만 한 군집은 내부 거리 넓고 품질 낮음 → 최적이 아님
- **k = 3**
  - 평균: ~0.588
  - 0번 군집이 너무 퍼져 있음
- **k = 4 (정답)**
  - 평균: ~0.65
  - **군집별 균형이 좋고, 각 군집의 실루엣 분포도 균일함**

따라서 이 데이터에서는 **k=4가 가장 적절**한 군집 개수로 판단

```
from sklearn.datasets import load_iris
iris=load_iris()
visualize_silhouette([ 2, 3, 4, 5 ], iris.data)
```



make_blobs를 통해 군집화를 위한 4개의 군집 중심의 500개 2차원 데이터 세트 생성

```
from sklearn.datasets import make_blobs
from sklearn.metrics import silhouette_samples, silhouette_score

# make_blobs로 데이터 생성 (4개의 군집)
X, y = make_blobs(n_samples=500, n_features=2, centers=4,
                  cluster_std=1, center_box=(-10.0, 10.0),
                  shuffle=True, random_state=1)

# 군집 개수 2~5 시각화
visualize_silhouette([2, 3, 4, 5], X)
```

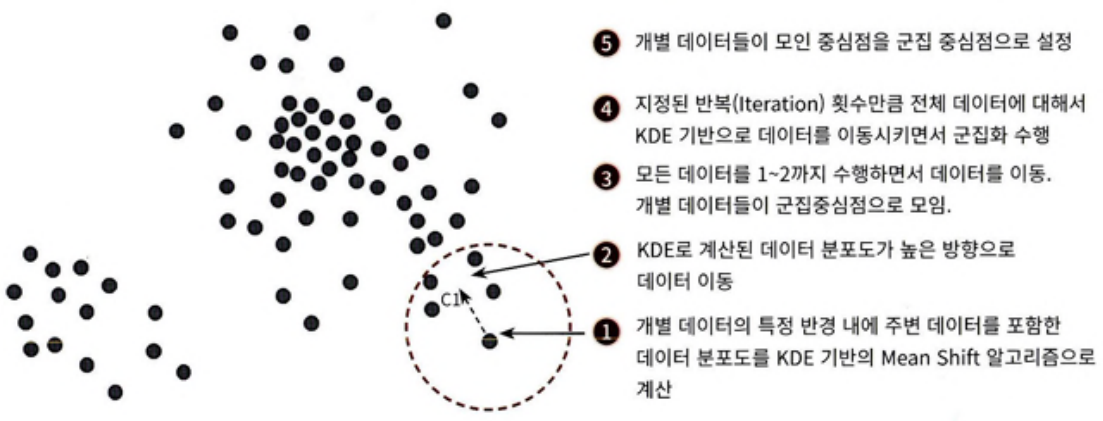
03 평균 이동 (Mean Shift)

평균 이동 개요

- K-평균처럼 중심점을 이동시키며 군집화를 수행하는 알고리즘
- K-평균: 군집에 속한 데이터의 **평균 위치(center of mass)** 로 이동
- 평균 이동: 데이터가 **가장 밀집된 방향(확률 밀도 최대 지점)** 으로 중심을 이동
- 군집 개수를 **사용자가 지정하지 않아도 됨**
- 데이터 분포 기반으로 자연스럽게 군집 개수 결정됨

확률 밀도 함수(PDF)를 이용한 중심 탐색 ... KDE

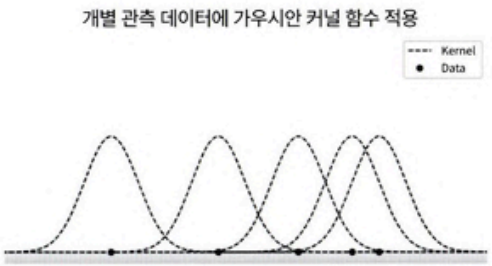
- 평균 이동은 데이터 밀집도를 기반으로 중심을 찾기 때문에 **확률 밀도 함수(PDF)** 를 활용
- 데이터가 가장 많이 모인 지점(확률 밀도 peak)을 군집 중심으로 사용
- PDF 추정을 위해 **KDE(Kernel Density Estimation)** 사용

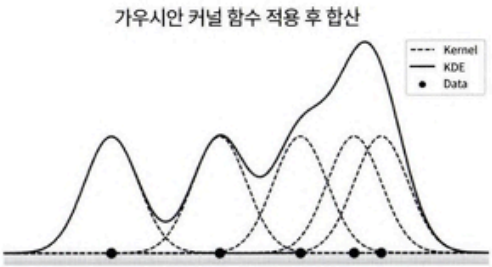


- 커널 함수(가우시안)를 데이터에 적용해 **확률 밀도 함수를 부드럽게 추정하는 방법**
- 주요 개념:
  - 관측 데이터마다 커널 함수를 적용
  - 모든 값을 합한 뒤 데이터 개수로 평균 → KDE 생성
- PDF의 형태를 통해 데이터의 분포 특성(평균, 분산, 분포 형태 등)을 이해할 수 있음
- KDE를 통해 **데이터가 가장 높은 밀도를 갖는 방향을 찾고 중심을 이동함**

KDE의 대역폭(bandwidth, h)의 의미

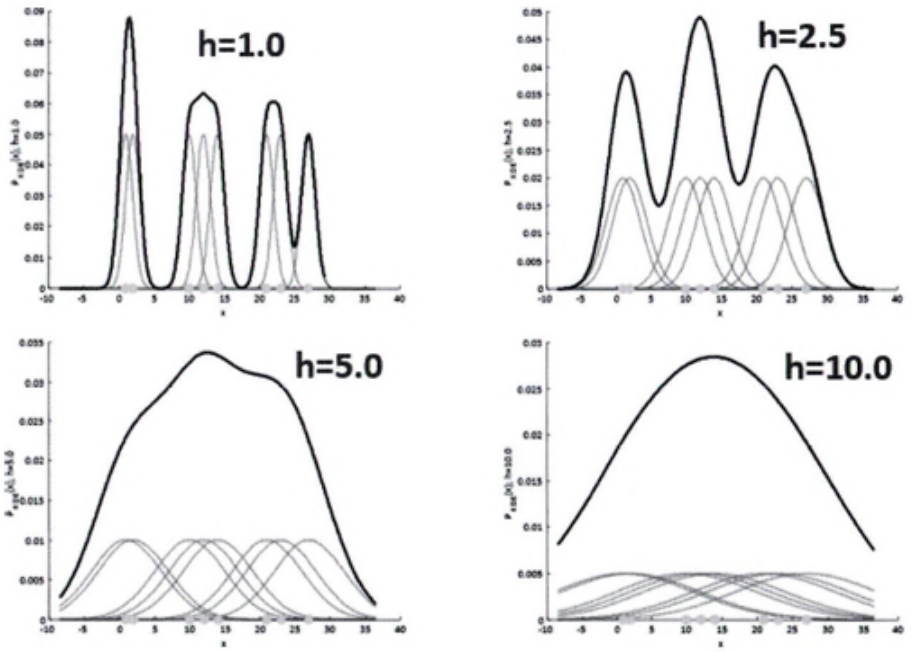
- h는 KDE를 평활화(smoothing)하는 정도를 결정하는 파라미터
- **h가 작을수록**
  - KDE가 뾰족하고 변동성이 커짐
  - 과적합(overfitting) 위험 ↑
- **h가 클수록**
  - KDE가 지나치게 부드러워져 단순화됨
  - 과소적합(underfitting) 위험 ↑
- 평균 이동에서 군집 개수는 **h 값에 의해 결정됨**
  - h ↑ → 군집 개수 ↓
  - h ↓ → 군집 개수 ↑





KDE는 다음과 같은 커널 함수식으로 표현됩니다. 다음 식에서 K는 커널 함수, x는 확률 변숫값, xi는 관측값, h는 대역폭(bandwidth)입니다.

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




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

clusterDF["meanshift_label"] = cluster_labels
centers = meanshift.cluster_centers_
unique_labels = np.unique(cluster_labels)

markers = ["o", "s", "^", "x", "*"]  # 5개 마커

plt.figure(figsize=(7, 5))

for label in unique_labels:
    label_cluster = clusterDF[clusterDF["meanshift_label"] == label]
    center_x_y = centers[label]

    # 군집 산점도
    plt.scatter(
        x=label_cluster["ftr1"],
        y=label_cluster["ftr2"],
        edgecolor="k",
        marker=markers[label]
    )

    # 중심점(큰 회색)
    plt.scatter(
```

```
        x=center_x_y[0],
        y=center_x_y[1],
        s=200,
        color="gray",
        alpha=0.9,
        marker=markers[label]
    )

    # 중심점(번호 표시)
    plt.scatter(
        x=center_x_y[0],
        y=center_x_y[1],
        s=70,
        color="white",
        edgecolor="k",
        marker=f"${label}$"
    )

plt.title("Mean Shift Clustering")
plt.show()

# ================================
# 타깃 vs 예측 군집 분포 보기
# ================================
print(clusterDF.groupby("target")["meanshift_label"].value_counts())
```

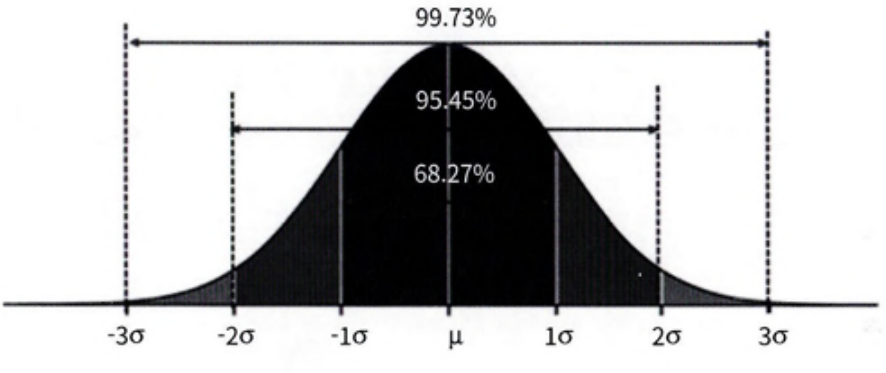
04 GMM (Gaussian Mixture Model)

GMM(Gaussian Mixture Model) 개요

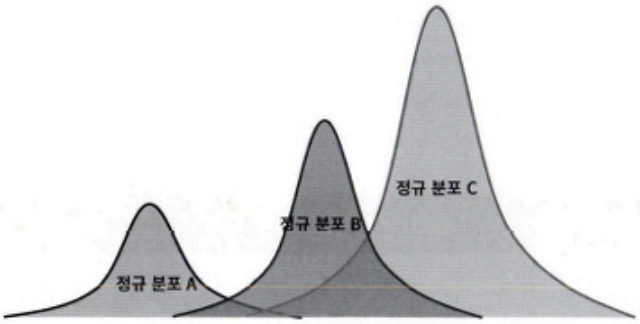
- GMM은 데이터가 **여러 개의 가우시안 분포(Gaussian Distribution)** 가 섞여 있다고 가정하고 군집화하는 방법
- 각 군집은 하나의 **정규 분포(Normal Distribution)** 로 표현됨
- 전체 데이터는 여러 개의 정규 분포가 합쳐진 형태로 구성된다고 보는 방식

가우시안(정규) 분포 Basics

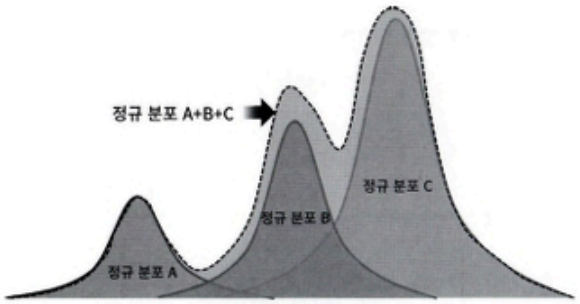
- 평균 μ를 중심으로 좌우 대칭을 이루는 **종(bell) 모양의 연속 확률 분포**
- 주요 특징:



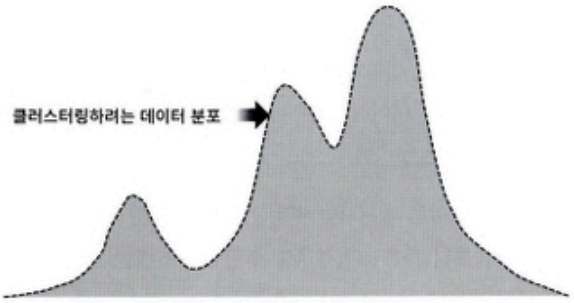
- μ ± 1σ: 전체 데이터의 **68.27%**
- μ ± 2σ: 전체 데이터의 **95.45%**
- μ = 0, σ = 1인 분포는 **표준 정규 분포(Standard Normal Distribution)**



이 세 개의 정규 분포를 합치면 다음 형태가 될 것입니다.



군집화를 수행하려는 실제 데이터 세트의 데이터 분포도가 다음과 같다면 쉽게 이 데이터 세트가 정규 분포 A, B, C가 합쳐서 된 데이터 분포도임을 알 수 있습니다.

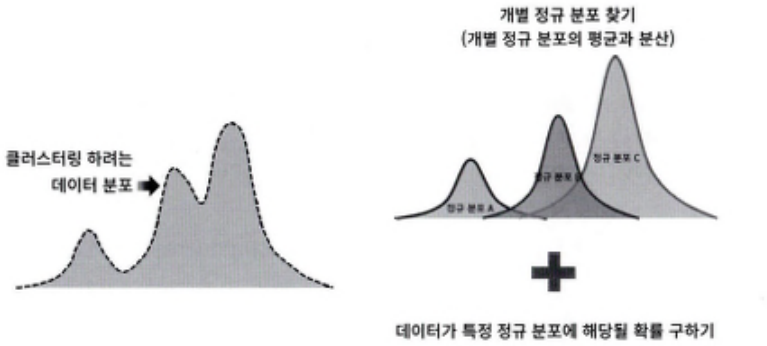


- 데이터 세트는 여러 개의 정규 분포 A, B, C...가 합쳐진 형태라고 가정
- 목표

  → 혼합된 데이터에서 **각각의 정규 분포를 추출하고**,

  → 개별 데이터가 어떤 정규 분포(=어떤 군집)에 속할지 확률적으로 판단하는 것

GMM의 모수(parameter) 추정



GMM은 다음 2 가지 모수를 추정해야 군집화를 수행할 수 있음

1. **각 가우시안 분포의 평균(μ)과 분산(σ^2)**
2. **각 데이터가 특정 가우시안 분포(군집)에 속할 확률**

- 즉, GMM은 **확률적 군집화(soft clustering)** 를 수행

  → K-means처럼 딱 한 군집에 속함이 아니라,

  각 군집에 속할 확률을 계산함

EM(Expectation–Maximization) 알고리즘

- GMM은 모수 추정을 위해 **EM 알고리즘**을 사용
- EM 알고리즘 과정
  - **E-step:** 각 데이터가 각 정규 분포에 속할 확률 계산
  - **M-step:** 확률을 기반으로 평균·분산·가중치를 다시 계산
- 반복 수행하여 최적의 가우시안 혼합 모델을 만듦
- 사이킷런의 `GaussianMixture` 클래스가 이 과정을 포함하고 있어 직접 구현할 필요 없음

GMM(확률 기반) vs K-평균(거리) 붓꽃 데이터 셋트 군집화 (실습)

```
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

# 데이터 로드
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
```

```
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 irisDF의 '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)
```

```
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)
```

GMM vs K-평균

KMeans 특성

- KMeans는 **원형(구형) 형태로 분포된 데이터**에서 군집화 성능이 높음
- `cluster_std`를 작게 설정한 `make_blobs()` 데이터를 사용하면 데이터가 원형 형태로 잘 모이므로 KMeans가 효과적
- 그러나 **길쭉한 타원형(anisotropic)** 데이터의 경우 원형 거리 중심 기반이라 군집화가 부정확해짐
- **= 방향성이 있는 분포에 취약**

문제 상황

- `make_blobs()`로 생성한 데이터를 특정 행렬로 변환해 **타원형 분포**로 재배치
- 이를 시각화하기 위해 `visualize_cluster_plot()` 함수를 사용

- **clusterobj**: KMeans 또는 GaussianMixture의 fitted 객체 (make_blobs 시각화일 경우 None)
- **dataframe**: feature + label을 포함한 DataFrame
- **label_name**: 표시할 레이블 컬럼명
- **iscenter**: 군집 중심을 시각화할지 여부 (GMM은 center 없음 → False)

KMeans 적용 결과

- 타원형 데이터에서 KMeans는 **원형 기준으로 그룹화**해 방향성 있는 군집을 제대로 반영하지 못함
- 일부 target 값과 군집 결과가 불일치

GMM(Gaussian Mixture Model)의 장점

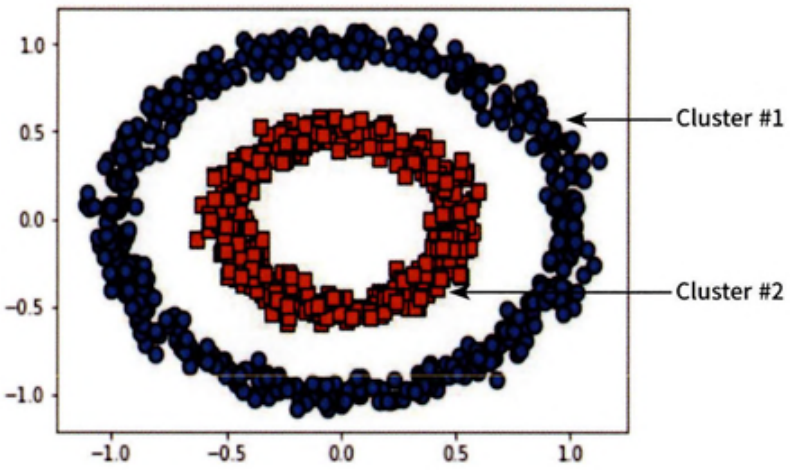
- GMM은 **타원형 분포**나 **방향성이 강한 데이터**에서도 군집화가 뛰어남
- 확률 모델 기반으로 각 클러스터를 Gaussian으로 가정하여 더 유연하게 군집화
- 군집 중심을 제공하지 않으므로 시각화 시 중심점 표시 불가
- KMeans에 비해 **계산 비용은 높지만 정확도는 높음**

성능 비교

- KMeans: 일부 군집이 target과 일치하지 않음
- GMM: 대부분의 군집이 target과 정확하게 매칭됨
- 결론: **데이터 형태가 원형이 아닐 경우 GMM이 더 적합**

05 DBSCAN

DBSCAN 개요



- DBSCAN(Density Based Spatial Clustering of Applications with Noise)
- **밀도 기반 군집화 알고리즘**
- K-평균, 평균 이동, GMM이 어려워하는 **기하학적으로 복잡한 데이터 분포**(원형, 비선형 분포)도 효과적으로 군집화 가능
- 2가지 핵심 파라미터
  - **eps(ε):** 반경(Neighborhood radius)
  - **min_samples:** 핵심 포인트가 되기 위해 필요한 최소 포인트 개수(자기 자신 포함)
- 포인트 유형
  - **Core Point:** 이웃이 min_samples 이상
  - **Neighbor Point:** eps 반경 안의 이웃 포인트
  - **Border Point:** Core Point와 인접하지만 min_samples 조건은 만족하지 못함
  - **Noise Point:** Core Point 인접도 아니고 최소 개수 미달 → 잡음

DBSCAN 동작 방식 예시

- P1~P12 데이터가 있을 때 min_samples = 6 가정
- P1: 이웃 7개 포함 → Core Point
- P2: 이웃 6개 포함 → Core Point
- Core Point 간은 연결되어 군집을 확장함
- P3: 이웃 적지만 Core Point(P2)와 연결 → Border Point
- P5: 이웃 적고 Core Point와 연결도 없음 → Noise Point
- DBSCAN은 이렇게 Core Point들을 연결하며 군집을 형성하고, Border/Noise를 구분

DBSCAN 주요 파라미터

- **eps:** 반경. ↑ 증가 → 더 많은 이웃 포함 → Noise 감소
- **min_samples:** 최소 이웃 수. ↑ 증가 → 더 촘촘해야 Core Point → Noise 증가
- 군집 개수는 자동 결정되므로 "군집 개수 설정"은 무의미

DBSCAN 적용: Iris 데이터 (실습)

- eps=0.6, min_samples=8 사용
- 결과:
  - 군집 레이블: 0, 1, **–1(Noise)**
  - Iris 데이터는 실제 target 3개지만 DBSCAN은 2개 군집 + noise로 나뉨

- PCA 변환 후 시각화:
  - Noise 점이 명확히 드러남

eps 변경 효과

- eps=0.6 → Noise 많음
- eps=0.8 → 반경 증가 → Noise 감소(–1 라벨 3개뿐)

min_samples 변경 효과

- min_samples=16 → Core Point 줄어듦 → Noise 증가

06 군집화 실습 - 고객 세그먼테이션

고객 세그먼테이션 개요

- 고객 세그먼테이션(Customer Segmentation)은 특정 기준으로 고객을 분류하는 기법
- CRM 및 타깃 마케팅의 핵심 기반
- 분류 기준 예
  - 인구통계 정보(지역, 성별, 결혼 여부, 소득 등)
  - 구매 행동 기반 정보(구매 빈도, 구매 금액 등 → **더 중요**)
- 기업 입장에서 핵심 기준: **매출 기여도**
- 목적: 고객 유형에 맞는 **맞춤형 마케팅** 추진(VIP 상품 추천, 프로모션 안내 등)

RFM 기법

- RFM은 고객 분석의 대표적인 기준
- 구성 요소
  - **Recency (R):** 가장 최근 구매 후 경과 일수
  - **Frequency (F):** 구매 횟수
  - **Monetary (M):** 총 구매 금액
- 이번 실습에서는 RFM 기반으로 고객 군집화 수행

실습

데이터 로드 및 기본 정보

- 데이터: Online Retail.xlsx (제품 주문 데이터)
- 주요 컬럼:

    - InvoiceNo: 주문번호 ('C'로 시작 시 취소)
    - StockCode: 제품코드
    - Description: 제품 설명
    - Quantity: 주문 수량
    - InvoiceDate: 주문일
    - UnitPrice: 단가
    - CustomerID: 고객 ID
    - Country: 고객 국가
- 로드 후 info() 확인 → CustomerID에 Null 값이 많음 (13.5만 건)

정제 작업

- Null 데이터 제거 → 특히 CustomerID가 Null인 row 삭제
- 오류 데이터 제거
    - Quantity 혹은 UnitPrice 가 0 이하인 값 제거
    - 'C'로 시작하는 취소 주문도 자연스럽게 제거됨 (Quantity < 0 처리 시 포함)
- 국가 기준 필터링 → **United Kingdom만 사용**
- 정제 후 데이터 개수: **541,909 → 397,884건**

RFM 계산을 위한 데이터 가공

- 새로운 칼럼 생성:
    - `sale_amount = Quantity × UnitPrice`
- CustomerID를 int 형으로 변환
- 고액/고빈도 주문 고객이 일부 존재 → 데이터 왜곡 가능성 높음
- InvoiceNo + StockCode 조합은 거의 유일한 key 수준 (1:1에 가까움)

고객 단위 데이터 생성(Groupby)

- RFM 구성에 필요한 aggregation:
    - **Recency:** InvoiceDate의 max() → 이후 날짜 차이로 변환
    - **Frequency:** InvoiceNo의 count()
    - **Monetary:** sale_amount의 sum()
- groupby(CustomerID).agg({...}) 활용하여 일괄 처리
- Recency 계산 시 기준 날짜:

- 실제 데이터 범위: 2010-12-01 ~ 2011-12-09
- 기준일(today)은 **2011-12-10**으로 설정
- Recency = 기준일 - 마지막 주문일 + 1일

RFM 변수의 분포 확인

- Recency, Frequency, Monetary 모두 **심한 skewed distribution(왜곡 분포)**
- 주요 특징:
  - Frequency, Monetary의 max 값이 매우 크며 일부 고객이 압도적으로 높은 값 보유
  - 중위(mean)와 평균(mean) 차이가 매우 큼 → long-tail 분포
- describe() 결과:
  - Recency 평균 92.7 / median 51 → 큰 차이
  - Frequency 평균 90.3 / 75%값이 99 → max 7847 때문에 왜곡
  - Monetary 평균 1864 / 75%값 1576 → max 259,657 때문에 왜곡

군집화 문제점 및 전처리 필요성

- 데이터 왜곡이 심한 상태에서 K-means 적용 시:
  - 군집이 한쪽으로 몰리는 현상 발생
  - 중심점(k-centroids)의 분리가 어려워짐
- 해결책 → StandardScaler 등 표준화 필요