11장
클러스터링

11장 클러스터링

11.1 클러스터링이란

11.2 클러스터링 알고리즘 유형

11.1 클러스터링이란

- **클러스터링이란**
  - 클러스터링은 어떤 데이터들이 주어졌을 때 특성이 비슷한 데이터끼리 묶어 주는 머신 러닝 기법
  - 머신 러닝 알고리즘에 딥러닝을 적용한다면 성능이 더 향상될 수 있음
  - 이 장에서는 그 방법을 배워 보자

11.2 클러스터링 알고리즘 유형

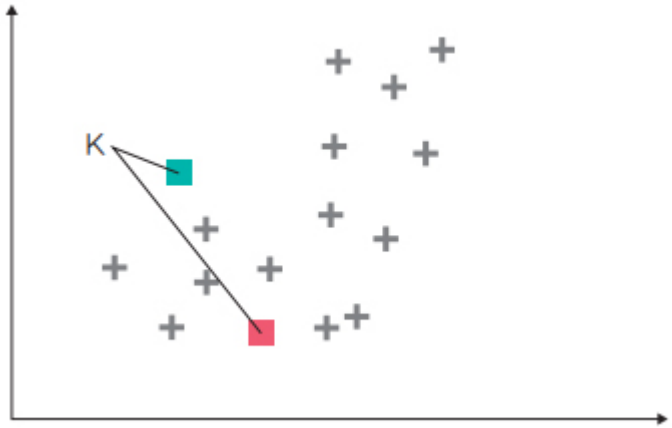
- **K-평균 군집화**

  K-평균 군집화 알고리즘 원리

  - K-평균 군집화 알고리즘의 학습 원리는 다음과 같음

  1. 클러스터 중심인 중심점을 구하기 위해 임의의 점 K(여기에서 K=2)를 선택

▼ **그림 11-1 임의의 점 K 선택**

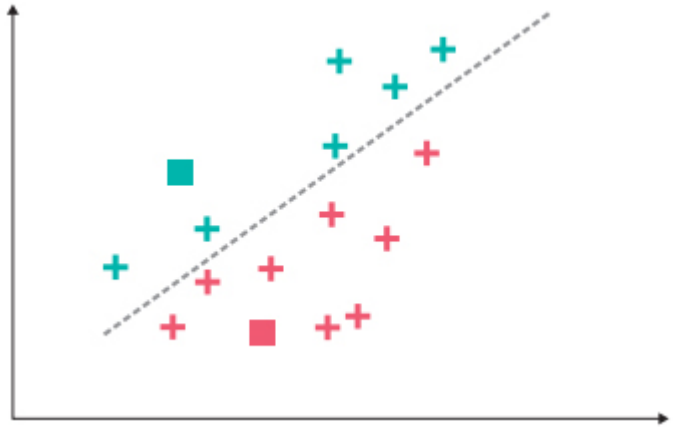


11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**

  2. 각 중심에 대한 거리를 계산하여 각 데이터를 가장 가까운 클러스터에 할당

▼ **그림 11-2 클러스터에 할당**



11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**

  3. 할당된 데이터 평균을 계산하여 새로운 클러스터 중심을 결정

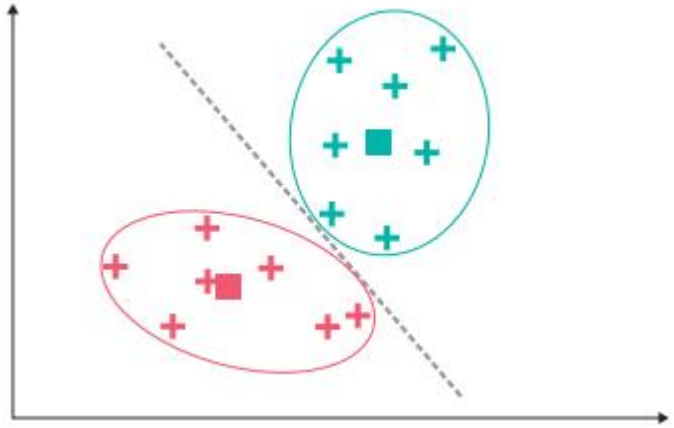
▼ **그림 11-3 새로운 중심 결정**

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**

  4. 클러스터 할당이 변경되지 않을 때까지 **2**~**3**을 반복

▼ **그림 11-4 최종 클러스터 구성**



11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - K-평균 군집화를 대략적으로 살펴보았으므로 이제 파이토치에서 사용하는 방법을 예제로 확인해 보자
  - 미리 말해 두지만 파이토치는 딥러닝을 위한 프레임워크이기 때문에 머신 러닝 기법들을 간편하게 사용할 수 있는 방법들을 제공하지 않음
  - 파이토치에서는 머신 러닝 알고리즘이 수행해야 할 일들을 개발자가 일일이 함수로 구현해 주어야 함
  - 이러한 과정이 번거롭다면 간편하게 사이킷런(scikit-learn) 라이브러리를 사용해야 함
  - 파이토치에서 머신 러닝 기법에 대한 사용 방법을 알아보는 이유는 여러분이 인터넷이나 논문을 통해 파이토치로 구현된 머신 러닝 알고리즘을 마주했을 때 부담 없이 코드를 받아들일 수 있도록 돕기 위해서임

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**

  파이토치로 K-평균 군집화 예제 구현하기

  - 파이토치에서 K-평균 군집화 알고리즘을 사용하여 데이터를 분류하는 방법을 살펴보자
  - 먼저 K-평균 군집화를 사용하기 위한 라이브러리를 설치해야 함
  - 아나콘다 프롬프트에서 다음 명령을 실행

    ```
    > pip install kmeans_pytorch
    ```

  - kmeans_pytorch 라이브러리는 파이토치에서 K-평균 군집화를 손쉽게 구현할 수 있도록 제공
  - 파이토치에서 모든 유형의 머신 러닝을 사용할 수 있는 라이브러리가 있는 것은 아님

```
[ ] #클러스터링

    !pip install kmeans_pytorch
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Collecting kmeans_pytorch
  Downloading kmeans_pytorch-0.3-py3-none-any.whl (4.4 kB)
Installing collected packages: kmeans-pytorch
Successfully installed kmeans-pytorch-0.3
```

```
import pandas as pd
from sklearn.model_selection import train_test_split  #
import torch
from kmeans_pytorch import kmeans, kmeans_predict   # k 평균 군집화
import matplotlib.pyplot as plt
```

```
[ ] from google.colab import files
    file_uploaded=files.upload()
```

파일 선택 선택된 파일 없음 Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.

Saving iris.csv to iris.csv

```
df=pd.read_csv('iris.csv')
print(df)
```

```
      Id  SepalLengthCm  SepalWidthCm  PetalLengthCm  PetalWidthCm  ₩
0      1            5.1           3.5            1.4           0.2
1      2            4.9           3.0            1.4           0.2
2      3            4.7           3.2            1.3           0.2
3      4            4.6           3.1            1.5           0.2
4      5            5.0           3.6            1.4           0.2
..   ...            ...           ...            ...           ...
145  146            6.7           3.0            5.2           2.3
146  147            6.3           2.5            5.0           1.9
147  148            6.5           3.0            5.2           2.0
148  149            6.2           3.4            5.4           2.3
149  150            5.9           3.0            5.1           1.8
```

```
                Species
0           Iris-setosa
1           Iris-setosa
2           Iris-setosa
3           Iris-setosa
4           Iris-setosa
..                  ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - 예제에서 사용하는 데이터는 아이리스 데이터셋
  - 다음 URL에서 데이터셋을 내려받을 수 있음

    https://www.kaggle.com/saurabh00007/iriscsv?select=Iris.csv

코드 11-2 데이터셋 불러오기

```
df = pd.read_csv('../chap11/data/iris.csv')
df.info() ------ 데이터셋에 대한 전반적인 정보를 출력
print('----------------------------------------')
print(df) ------ 아이리스 데이터셋의 데이터 출력
```

  - df.info()를 통해 확인하고 싶은 내용은 데이터 타입
  - 데이터 타입이 'object'라면 'float64'로 바꾸어야 하기 때문에 사전 확인이 필요함

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - 다음은 불러온 데이터셋에 대한 정보

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 6 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Id             150 non-null    int64
 1   SepalLengthCm  150 non-null    float64
 2   SepalWidthCm   150 non-null    float64
 3   PetalLengthCm  150 non-null    float64
 4   PetalWidthCm   150 non-null    float64
 5   Species        150 non-null    object
```

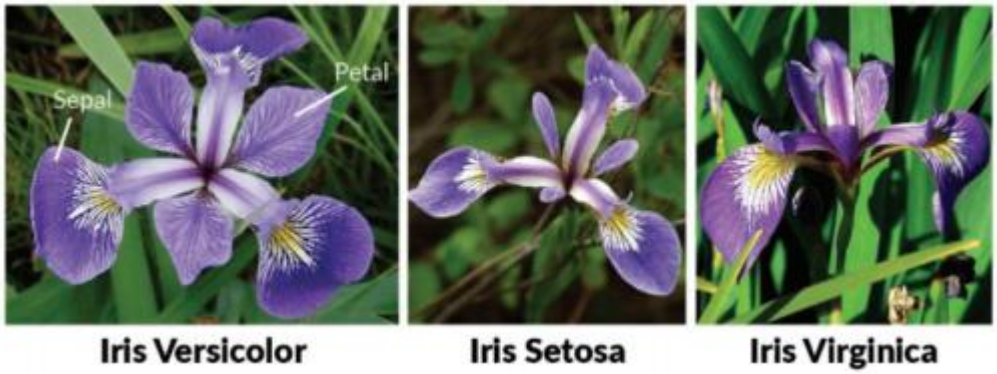
11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**

  아이리스(붓꽃) 데이터셋

  - 아이리스(붓꽃) 데이터셋은 꽃잎의 너비와 길이 등을 측정한 데이터이며 150개의 레코드로 구성
  - 아이리스 꽃은 다음 그림과 같으며 프랑스의 국화로도 알려져 있음

▼ 그림 11-7 아이리스 꽃(출처: https://www.kaggle.com/alexisbcook/distributions)



11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - 'object'라는 데이터 타입을 갖는 Species 칼럼은 숫자가 아닌 단어로 구성
  - 단어는 꽃잎의 너비와 길이에 따라 아이리스(붓꽃)의 세 가지 범주를 나타냄
  - 이와 같이 단어를 숫자로 바꾸어 주는 것을 워드 임베딩이라고 하며, 이를 위한 다양한 방법이 있지만 여기에서는 get_dummies()를 사용

코드 11-3 워드 임베딩

```
data = pd.get_dummies(df, columns=['Species'])
data
```

```
                Species
0       Iris-setosa
1       Iris-setosa
2       Iris-setosa
3       Iris-setosa
4       Iris-setosa
..                  ...
145  Iris-virginica
146  Iris-virginica
147  Iris-virginica
148  Iris-virginica
149  Iris-virginica

[150 rows x 6 columns]
```

```
[ ] data=pd.get_dummies(df,columns=['Species'])   # dummies()
    data
```

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species_Iris-setosa	Species_Iris-versicolor	Species_Iris-virginica
0	1	5.1	3.5	1.4	0.2	1	0	0
1	2	4.9	3.0	1.4	0.2	1	0	0
2	3	4.7	3.2	1.3	0.2	1	0	0
3	4	4.6	3.1	1.5	0.2	1	0	0
4	5	5.0	3.6	1.4	0.2	1	0	0
...	...	...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	0	0	1
146	147	6.3	2.5	5.0	1.9	0	0	1
147	148	6.5	3.0	5.2	2.0	0	0	1
148	149	6.2	3.4	5.4	2.3	0	0	1
149	150	5.9	3.0	5.1	1.8	0	0	1

150 rows × 8 columns

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - 다음은 워드 임베딩이 적용된 데이터셋 결과

	Id	SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species_Iris-setosa	Species_Iris-versicolor	Species_Iris-virginica
0	1	5.1	3.5	1.4	0.2	1	0	0
1	2	4.9	3.0	1.4	0.2	1	0	0
2	3	4.7	3.2	1.3	0.2	1	0	0
3	4	4.6	3.1	1.5	0.2	1	0	0
4	5	5.0	3.6	1.4	0.2	1	0	0
...	...	...	...	...	...	...	...	...
145	146	6.7	3.0	5.2	2.3	0	0	1
146	147	6.3	2.5	5.0	1.9	0	0	1
147	148	6.5	3.0	5.2	2.0	0	0	1
148	149	6.2	3.4	5.4	2.3	0	0	1
149	150	5.9	3.0	5.1	1.8	0	0	1

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - 이제 훈련과 테스트 용도로 데이터셋을 분리할 텐데 예제에서는 사이킷런을 사용
  - 데이터 분리는 사이킷런뿐만 아니라 파이토치의 random_split()을 사용할 수도 있음
  - random_split()을 사용할 경우 데이터셋의 인덱스를 텐서 형태로 반환하기 때문에 인덱스가 아닌 데이터를 이용해야 하는 이번 예제와 맞지 않아 사이킷런을 사용

코드 11-4 데이터셋 분리

```
from sklearn.model_selection import train_test_split

x, y = train_test_split(data, test_size=0.2, random_state=123)
```

```
[ ] from sklearn.model_selection import train_test_split

    x,y=train_test_split(data, test_size=0.2, random_state=123)
```

```
[ ] if torch.cuda.is_available():
        device = torch.device('cuda:0')
    else:
        device = torch.device('cpu')
```

```
[ ] from sklearn.preprocessing import StandardScaler

    scaler=StandardScaler()
    x_scaled=scaler.fit(data).transform(x)
    y_scaled=scaler.fit(y).transform(y)
```

Parameter

- arrays : 분할시킬 데이터를 입력
- test_size : 테스트 데이터셋의 비율(float)이나 갯수(int)
- train_size : 학습 데이터셋의 비율(float)이나 갯수(int)
- random_state : 데이터 분할시 셔플이 이루어지는데 이를 위한 시드값 (int나 RandomState로 입력)
- shuffle : 셔플여부설정 (default = True)
- stratify : 지정한 Data의 비율을 유지.

```
from sklearn.model_selection import train_test_split

train_test_split(arrays, test_size, train_size, random_state, shuffle, stratify
```

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - 워드 임베딩이 적용된 데이터셋의 결과를 보면 데이터 값이 다양함(SepalLengthCm, SepalWidthCm, PetalLengthCm, PetalWidthCm 칼럼은 1~6의 값을 가지며, Species는 0~1의 값을 갖음)
  - 이러한 값의 범위가 평균은 0, 분산은 1이 되도록 조정
  - 이러한 것을 특성 스케일링(feature scaling)이라고 하며, 대표적인 방법으로는 표준화(standardization)와 정규화(normalization)가 있음
  - 이번 예제에서는 표준화를 사용

코드 11-6 특성 스케일링

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_scaled = scaler.fit(data).transform(x) ------①
y_scaled = scaler.fit(y).transform(y)
```

```
[ ] # 텐서로 변경하기
    x=torch.from_numpy(x_scaled)
    y=torch.from_numpy(y_scaled)
```

```
print(x.size())
print(y.size())
print(x)
```

```
torch.Size([120, 8])
torch.Size([30, 8])
tensor([[ 1.2817e+00,  1.8862e+00, -5.8776e-01,  1.3314e+00,  9.2206e-01,
         -7.0711e-01, -7.0711e-01,  1.4142e+00],
        [ 1.0277e+00,  1.8983e-01, -1.9762e+00,  7.0589e-01,  3.9617e-01,
         -7.0711e-01, -7.0711e-01,  1.4142e+00],
        [-1.0508e+00, -1.3854e+00,  3.3785e-01, -1.2275e+00, -1.3130e+00,
          1.4142e+00, -7.0711e-01, -7.0711e-01],
        [-1.7205e+00, -9.0068e-01,  1.0321e+00, -1.3413e+00, -1.3130e+00,
          1.4142e+00, -7.0711e-01, -7.0711e-01],
```

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - ① StandardScaler()는 각 특성의 평균을 0, 분산을 1로 변경하여 특성의 스케일을 변경

```
X_scaled = scaler.fit(data).transform(x)
                      ⓐ          ⓑ
```

ⓐ fit: 평균과 표준편차를 계산

ⓑ transform: 계산된 평균과 표준편차를 적용

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - 데이터셋에 대한 전처리가 완료되었기 때문에 훈련과 테스트 데이터셋에 대한 크기를 확인해 보자

  - 결과를 보면 훈련 데이터셋은 120개, 테스트 데이터셋은 30개로 분리
  - 또한, from_numpy()가 적용되었기 때문에 훈련과 테스트 데이터셋은 텐서로 변경된 상태

코드 11-8 훈련과 테스트 데이터셋 크기

```
print(x.size()) ------ 훈련 데이터셋에 대한 크기
print(y.size()) ------ 테스트 데이터셋에 대한 크기
print(x) ------ 텐서로 변경된 훈련 데이터셋의 데이터 확인
```

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - 다음은 훈련과 테스트 데이터셋의 크기에 대한 결과

```
torch.Size([120, 8])
torch.Size([30, 8])
tensor([[ 1.2817e+00,  1.8862e+00, -5.8776e-01,  1.3314e+00,  9.2206e-01, -7.0711e-01,
         -7.0711e-01,  1.4142e+00],
        [ 1.0277e+00,  1.8983e-01, -1.9762e+00,  7.0589e-01,  3.9617e-01,
         -7.0711e-01, -7.0711e-01,  1.4142e+00],
        [-1.0508e+00, -1.3854e+00,  3.3785e-01, -1.2275e+00, -1.3130e+00, 1.4142e+00,
         -7.0711e-01, -7.0711e-01],
        [-1.7205e+00, -9.0068e-01,  1.0321e+00, -1.3413e+00, -1.3130e+00, 1.4142e+00,
         -7.0711e-01, -7.0711e-01],
        [-2.8868e-01,  1.8983e-01, -1.9762e+00,  1.3724e-01, -2.6119e-01, -7.0711e-01,
         1.4142e+00, -7.0711e-01],
... 중간 생략 ...
```

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - 이제 K-평균 군집화를 적용

코드 11-9 K-평균 군집화 적용

```
num_clusters = 3 ------ 아이리스(붓꽃) 유형이 세 개라서 3으로 지정
cluster_ids_x, cluster_centers = kmeans(
    X=x, num_clusters=num_clusters, distance='euclidean', device=device
) ------ ①
```

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - ① K-평균 군집화 알고리즘에 대한 훈련은 kmeans()를 이용하며, 이때 사용되는 파라미터는 다음과 같음

```
cluster_ids_x, cluster_centers = kmeans(
    X=x, num_clusters=num_clusters, distance='euclidean', device=device)
    ⓐ            ⓑ                  ⓒ              ⓓ
```

    ⓐ 첫 번째 파라미터: 훈련 데이터셋

    ⓑ num_clusters: 클러스터 개수

  ⓒ distance: 클러스터를 구성하기 위해 데이터 간의 거리를 계산하는 방법

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - 학습 결과는 다음과 같은 의미를 갖음

```
center_shift=0.000000, iteration=4, tol=0.000100
```
ⓐ ⓑ ⓒ

ⓐ center_shift: 유클리드 거리 계산을 이용하여 반복적으로 클러스터 중심을 변경
이 과정은 더 이상 클러스터 중심을 옮길 필요가 없을 때까지 반복되며 'iteration' 파라미터에 의해 최종 반복 횟수가 출력

ⓑ iteration: 중지할 기준(tol)을 만족하면 학습을 멈추고 반복 횟수를 출력

ⓒ tol: tolerance의 약자로 학습을 중지할 기준이 되는 값

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - 이제 테스트 데이터셋을 이용하여 예측해 보자

코드 11-11 K-평균 군집화 예측

```
cluster_ids_y = kmeans_predict(
    y, cluster_centers, 'euclidean', device=device
) ------ 예측을 위해서는 kmeans_predict()를 사용
```

  - 다음은 K-평균 군집화에 대한 예측 결과

```
predicting on cpu..
```

  - 별다른 결과 없이 어떤 장치(GPU/CPU)를 사용하는지에 대해서만 출력

11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - 예측 결과를 그래프로 알아보기 전에 테스트 데이터셋에 대한 클러스터 ID 값을 먼저 살펴보자

코드 11-12 테스트 데이터셋에 대한 클러스터 ID

```
print(cluster_ids_y)
```

  - 다음은 테스트 데이터셋에 대한 클러스터 ID 값

```
tensor([0, 2, 2, 0, 1, 2, 0, 1, 1, 0, 2, 1, 0, 2, 2, 2, 1, 1, 0, 1, 1, 2, 1, 2, 1, 1,
1, 2, 2, 1])
```

  - ID 값은 훈련 데이터셋과 마찬가지로 0, 1, 2의 범주 값(앞에서 클러스터를 3으로 설정)으로 구성되어 있음

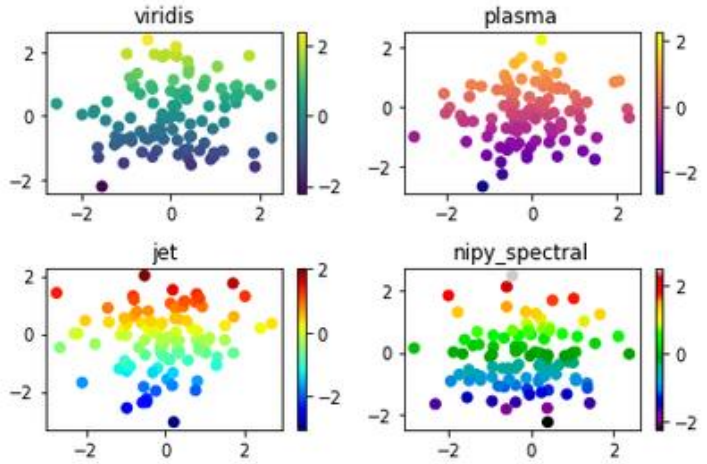
11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - 이제 마지막으로 예측 결과를 그래프로 살펴보자

코드 11-13 예측 결과 그래프로 확인

```
import matplotlib.pyplot as plt

plt.figure(figsize=(4, 3), dpi=160)
plt.scatter(y[:, 0], y[:, 1], c=cluster_ids_y, cmap='viridis', marker='x')
                                                    테스트 데이터셋에 적용된 클러스터 결과 출력
plt.scatter(
    cluster_centers[:, 0], cluster_centers[:, 1],
    c='white',
    alpha=0.6,
    edgecolors='black',
    linewidths=2
)

plt.tight_layout() ------ ①
plt.show()
```

```
plt.figure(figsize=(4, 3), dpi=160)
plt.scatter(y[:, 0], y[:, 1],c=cluster_ids_y, cmap='viridis', marker='x')

plt.scatter(
    cluster_centers[:, 0], cluster_centers[:, 1],
    c='white',
    alpha=0.6,
    edgecolors='black',
    linewidths=2
)

plt.tight_layout()
plt.show()
```



11.2 클러스터링 알고리즘 유형

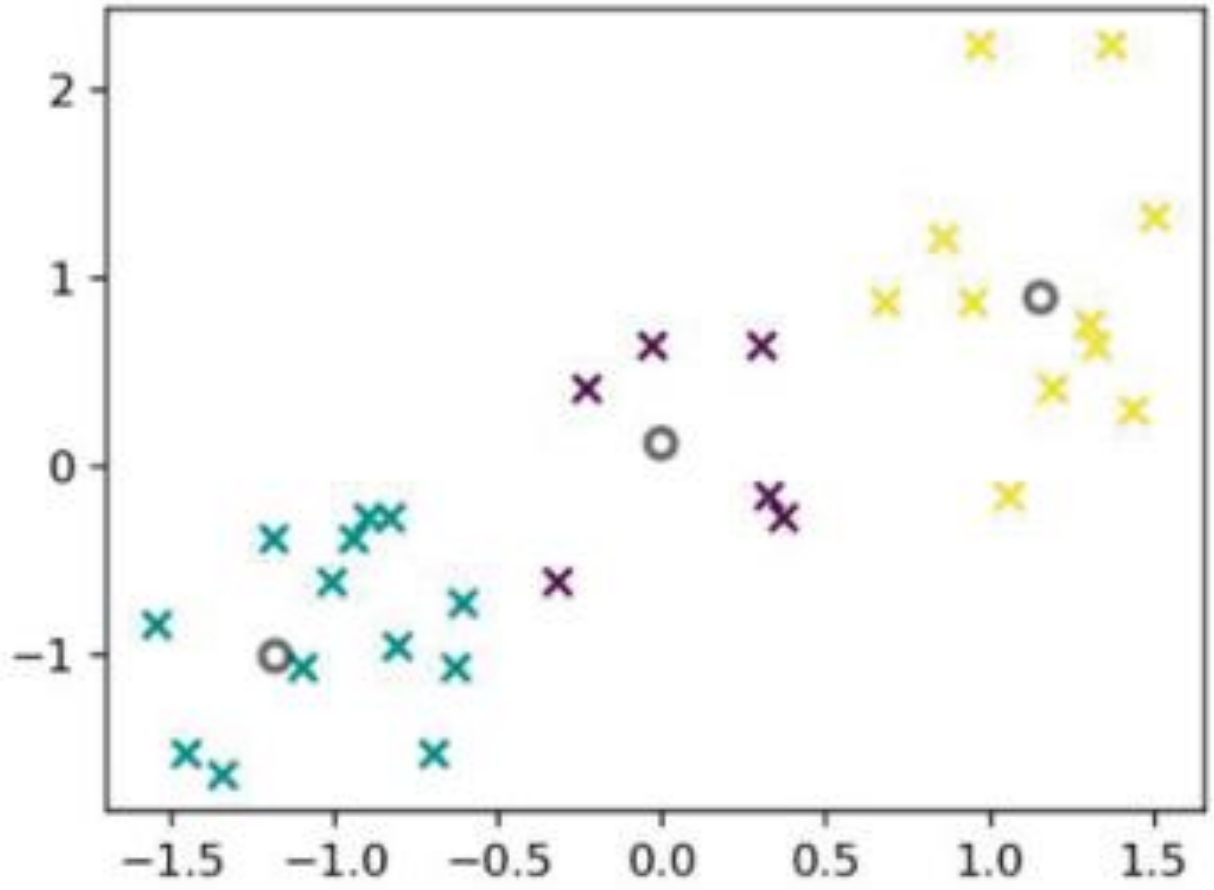
- **K-평균 군집화**
  - ① subplot을 사용하면 하나의 그래프가 다른 레이아웃에 넘어가는 경우가 종종 발생하는데, 이때 tight_layout()을 사용하면 그래프가 레이아웃에 자동으로 맞추어서 출력
  - 다음은 K-평균 군집화에 대한 예측을 그래프로 출력한 결과

```
[running kmeans]: 4it [00:03,  1.06it/s, center_shift=0.000000, iteration=4,
tol=0.000100]
```

>>> 주어진 데이터에 대해 k-means 알고리즘이 4번의 반복을 수행하였고, 중심점의 이동이 거의 없어서 알고리즘이 수행되었습니다. 이는 클러스터링 결과가 안정화되었음을 의미함.

11.2 클러스터링 알고리즘 유형

▼ 그림 11-9 K-평균 군집화 결과



11.2 클러스터링 알고리즘 유형

- **K-평균 군집화**
  - 데이터들을 하나의 클러스터로 묶기 위한 중심이 형성된 것을 확인할 수 있음
  - 이번 예제의 중요한 단점이 하나 있었음
  - 더 좋은 결과를 얻을 수 있는 하이퍼파라미터 변경이 매우 제한적이라는 것
  - 더 많은 하이퍼파라미터를 수정하여 성능을 향상하고 싶다면 kmeans_pytorch 라이브러리를 이용하는 것이 아닌, 클러스터를 형성하고 거리를 계산하는 부분들을 모두 개발자가 직접 함수로 구현해야 함
  - 매우 번거롭고 귀찮은 작업
  - 이와 같이 파이토치는 딥러닝을 위한 프레임워크
  - 머신 러닝 알고리즘을 적용하고 싶다면 파이토치가 아닌 사이킷런 같은 라이브러리를 사용하는 것이 편리함