인공신경망과딥러닝입문

Lecture 14. k-Means Clustering Algorithm

동덕여자대학교
데이터사이언스 전공
권 범

목차

01. k-Means Clustering Algorithm 소개

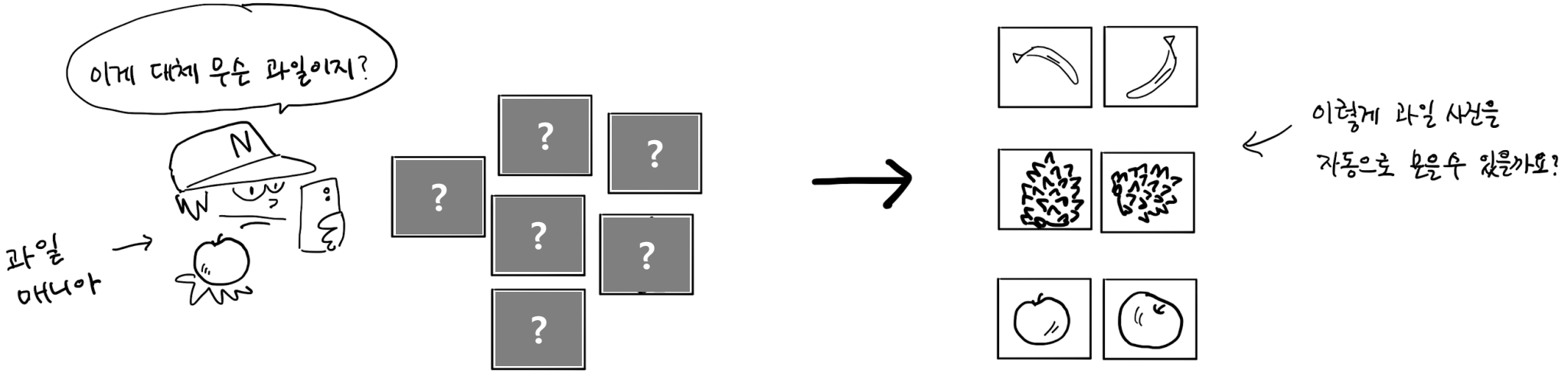
02. KMeans 클래스
03. 클러스터 중심
04. 최적의 k 찾기
05. 과일을 자동으로 분류하기
06. 마무리

01. k-Means Clustering Algorithm 소개

❖ 시작하기 전에 (1/2)

- 이전 수업 시간에 우리는 사과, 바나나, 파인애플 사진에 있는 각 픽셀의 평균값을 구해서 가장 가까운 사진을 골랐음
- 이 경우에는 사과, 바나나, 파인애플 사진임을 미리 알고 있었기 때문에 각 과일의 평균을 구할 수 있었음
- 하지만, 진짜 비지도 학습에서는 사진에 어떤 과일이 찍혀 있는지 알지 못함

이런 경우 어떻게 평균값을 구할 수 있을까?



[사진출처] 혼자 공부하는 머신러닝+딥러닝 (출판사: 한빛아카데미)

01. k-Means Clustering Algorithm 소개

❖ 시작하기 전에 (2/2)

- k-Means Clustering Algorithm이 평균값을 찾아 줌
- 이 평균값이 클러스터의 중심에 위치하기 때문에
  클러스터 중심(Cluster Center) 또는 센트로이드(Centroid)라고 부름

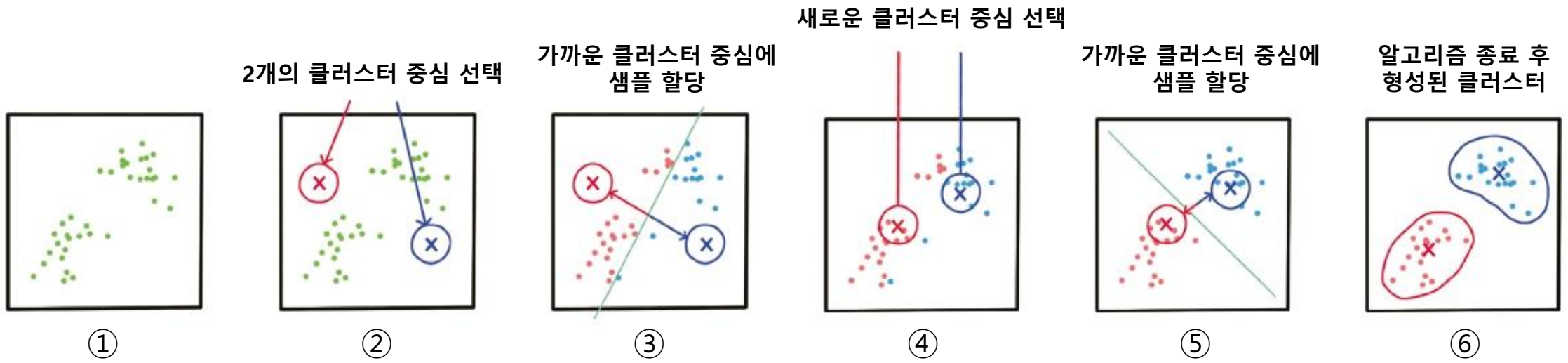
이번 수업 시간에는 k-Means Clustering Algorithm의 동작 원리를 이해하고 사과, 바나나, 파인애플을 구분하는 비지도 학습 모델을 만들어 보겠음

01. k-Means Clustering Algorithm 소개

❖ 동작 원리 (1/2)

군집 과정 요약

① 학습 데이터의 분포임
② 무작위로 k개의 클러스터 중심을 정함(본 예제에서는 k=2로 가정하였음)
③ 각 샘플에서 가장 가까운 클러스터 중심을 찾아 해당 클러스터의 샘플로 지정함
④ 클러스터에 속한 샘플의 평균값으로 클러스터 중심을 변경함
⑤ 클러스터 중심에 변화가 없을 때까지 ③으로 돌아가 반복함
⑥ 클러스터 중심에 변화가 없으면 알고리즘을 종료함



[사진출처] IT CookBook, 난생처음 인공지능 입문 (출판사: 한빛아카데미)

01. k-Means Clustering Algorithm 소개

❖ 동작 원리 (2/2)

“ k-Means Clustering Algorithm ”

**처음에는 랜덤하게 클러스터 중심을 선택하고 점차
가장 가까운 샘플의 중심으로 이동하는 비교적 간단한 알고리즘임**

이번 수업 시간에는 scikit-learn으로
k-Means Clustering 모델을 직접 만들어 보도록 하겠음

02. KMeans 클래스

01. k-Means Clustering Algorithm 소개
03. 클러스터 중심
04. 최적의 k 찾기
05. 과일을 자동으로 분류하기
06. 마무리

02. KMeans 클래스

❖ ① 실습 Dataset 준비하기 (1/3)

- 이전 수업 시간에서 사용했던 Fruits 360 데이터셋을 이번 수업 시간에도 사용하자

```
1  import os
2  import numpy as np
3  import matplotlib.pyplot as plt
4  import matplotlib.image as mpimg
5
6  cur_dir = os.getcwd()
7  fruit_list = ["Apple", "Banana", "Pineapple"]
8  fruit_npy = []
9  for fruit_name in fruit_list:
10   folder_name = cur_dir + "\\" + fruit_name
11   file_list = os.listdir(folder_name)
12   for file_name in file_list:
13     img = mpimg.imread(folder_name + "\\" + file_name)
14
15     R, G, B = img[:,:,0], img[:,:,1], img[:,:,2]
16     imgGray = 0.299 * R + 0.587 * G + 0.114 * B
17     imgGray = np.array(imgGray, dtype="int")
18     imgGray2 = 255 - imgGray
19     fruit_npy.append(imgGray2)
```

- ✓ getcwd( ) 함수:
  현재 작업 디렉토리(Current Working Directory) 문자열을 반환함
- ✓ listdir( ) 함수:
  지정한 디렉토리 내의 모든 파일과 디렉토리의 리스트를 반환함

02. KMeans 클래스

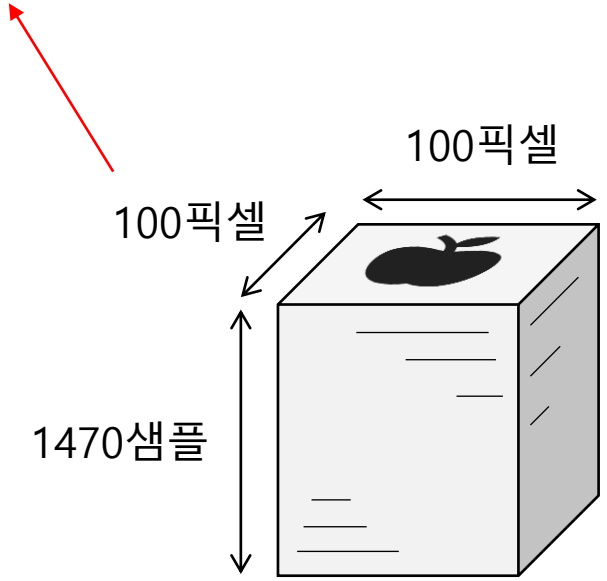
❖ ① 실습 Dataset 준비하기 (2/3)

- fruit_npy 배열의 shape 속성을 확인해 보자

```
1 fruit_npy = np.array(fruit_npy)
2 print(fruit_npy.shape)
```

실행결과

```
(1470, 100, 100)
```

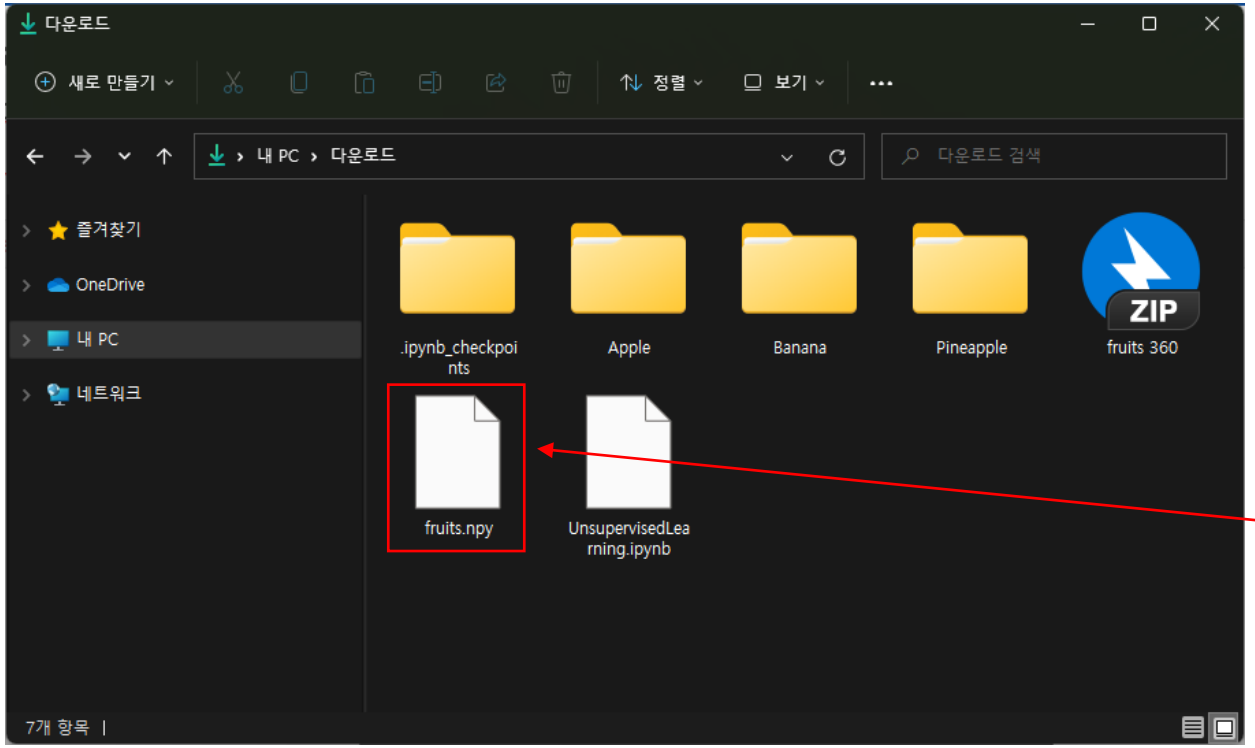


02. KMeans 클래스

❖ ① 실습 Dataset 준비하기 (3/3)

- fruit_npy 배열을 저장해 보겠음
- 넘파이 배열의 기본 저장 포맷인 npy 파일로 저장하면 됨

```
1 np.save("fruits.npy", fruit_npy)
```



현재 작업 중인 폴더에 저장된 것을 확인할 수 있음

02. KMeans 클래스

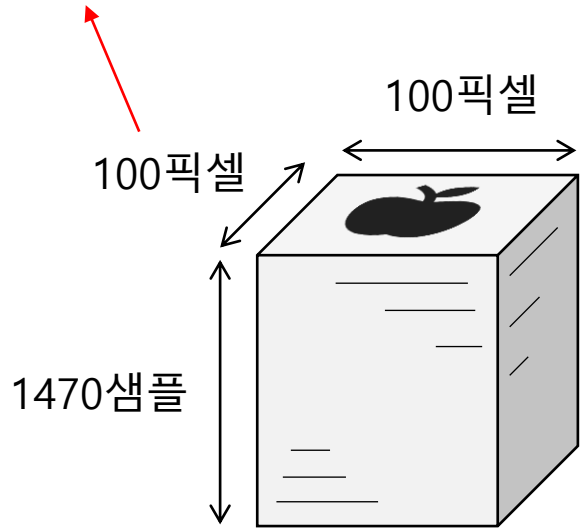
❖ ② npy 파일 로드하기

- 넘파이 np.load( ) 함수를 사용해 npy 파일을 읽어 넘파이 배열을 준비함

```
1 fruits = np.load("fruits.npy")
2
3 print(fruits.shape)
```

실행결과

```
(1470, 100, 100)
```



- ✓ fruits는 3차원 넘파이 배열임
- ✓ 이 배열의 첫 번째 차원(1470)은 샘플의 개수를 나타냄
- ✓ 두 번째 차원(100)은 이미지의 높이를 나타냄
- ✓ 세 번째 차원(100)은 이미지의 너비를 나타냄

k-Means Clustering Algorithm을 학습시키기 위해서는 배열의 크기를 변경해야 함

02. KMeans 클래스

❖ ③ 3차원 배열을 2차원 배열로 변경하기

- k-Means Clustering Algorithm을 학습시키기 위해서, (샘플 개수, 너비, 높이) 크기의 3차원 배열을 (샘플 개수, 너비×높이) 크기를 가진 배열로 변경함
- reshape( ) 메서드를 이용하면 쉽게 배열의 크기를 변경할 수 있음

```
1 fruits_2d = fruits.reshape(-1, 100*100)
2 print(fruits_2d.shape)
```

실행결과

```
(1470, 10000)
```

(샘플 개수, 너비, 높이)

(1470, 100, 100)

(샘플 개수, 너비×높이)

(1470, 10000)

02. KMeans 클래스

❖ ④ KMeans 클래스의 객체를 생성하고 학습시키기

- scikit-learn의 k-Means Clustering Algorithm은 sklearn.cluster 모듈 아래 KMeans 클래스에 구현되어 있음
- 이 클래스에 설정할 매개변수는 클러스터 개수를 지정하는 n_clusters임
- 이 클래스를 사용하는 방법도 다른 클래스들과 비슷함
- 다만 비지도 학습이므로 fit( ) 메서드에서 타겟 데이터를 사용하지 않음

```
1 from sklearn.cluster import KMeans
2
3 km = KMeans(n_clusters=3, init="random",
4             random_state=42)
5 km.fit(fruits_2d)
```

- ✓ 클러스터 개수 n_clusters를 3으로 지정하겠음
- ✓ Init 매개변수를 "random"으로 설정하면, 초기 클러스터 중심 설정을 위해서 데이터에서 랜덤하게 n_clusters(=3)개의 샘플을 선택함

KMeans 객체를 만들 때 왜 random_state를 지정하나요?

- ✓ k-Means Clustering Algorithm은 처음에 무작위로 k개의 클러스터 중심을 정함
- ✓ 즉, 무작위성이 주입되기 때문에 실행할 때마다 형성되는 클러스터가 달라질 수 있음
- ✓ 여기서는 실습 결과가 같도록 유지하기 위해서 random_state를 지정하지만, 실전에서는 필요하지 않음

02. KMeans 클래스

❖ ⑤ 군집 결과 확인하기 (1/2)

- 군집된 결과는 KMeans 클래스 객체의 labels_ 속성에 저장됨
- labels_ 배열의 길이는 샘플 개수와 같음
- 이 배열은 각 샘플이 어떤 레이블(Label)에 해당되는지 나타냄
- n_clusters=3으로 지정했기 때문에 labels_ 배열의 값은 0, 1, 2 중 하나임

```
1 print(km.labels_)
2 print(km.labels_.shape)
```

실행결과

```
[1 1 1 ... 1 1 1]
(1470,)
```

레이블값 0, 1, 2와 레이블 순서에는 어떤 의미도 없음

02. KMeans 클래스

❖ ⑤ 군집 결과 확인하기 (2/2)

- 실제 레이블 0, 1, 2가 어떤 과일 사진을 주로 모았는지 알아보려면 직접 이미지를 출력하는 것이 최선임
- 그 전에 레이블 0, 1, 2로 모은 샘플의 개수를 확인해 보자

```
1 print(np.unique(km.labels_, return_counts=True))
```

실행결과

```
(array([0, 1, 2], dtype=int32), array([490, 735, 245]))
```

✓ 첫 번째 클러스터(레이블 0)는 490개의 샘플을 모았음
✓ 두 번째 클러스터(레이블 1)는 735개의 샘플을 모았음
✓ 세 번째 클러스터(레이블 2)는 245개의 샘플을 모았음

각 클러스터가 어떤 이미지를 나타냈는지 그림으로 출력하기 위해 간단한 유틸리티 함수 draw_fruits( )를 만들어 보겠음

02. KMeans 클래스

❖ ⑥ draw_fruits( ) 함수 만들기 (1/2)

- (샘플 개수, 너비, 높이)의 3차원 배열을 입력 받아 가로로 10개씩 이미지를 출력시키겠음

```
1  def draw_fruits(arr, ratio=1):
2    n = len(arr)
3
4    rows = int(np.ceil(n/10))
5    cols = n if rows < 2 else 10
6
7    fig, axs = plt.subplots(rows, cols, figsize=(cols * ratio, rows * ratio),
8                            squeeze=False)
9
10   for j in range(rows):
11     for k in range(cols):
12       if j * 10 + k < n:
13         axs[j, k].imshow(arr[j * 10 + k],
14                          cmap="gray_r")
15
16       axs[j, k].axis("off")
17
18   plt.show()
```

✓ 샘플 개수에 따라 행과 열의 개수를 계산하고 figsize를 지정함
✓ figsize는 ratio 매개변수에 비례하여 커짐
✓ ratio 기본값은 1임

✓ squeeze의 기본값은 True임
✓ True로 설정되어 있으면 2차원 (1, 2)를 압축해서 (2,)로 만듦
✓ 이 경우 axs[j, k] 형태로 지정할 수 없음
✓ 코드의 통일성을 위해 2차원 형태로 axs를 반환할 수 있도록 squeeze를 False로 지정함

✓ 2중 for 반복문을 사용하여 첫 번째 행을 따라 이미지를 그림
✓ 그리고 두 번째 행의 이미지를 그리는 식으로 계속됨

02. KMeans 클래스

❖ ⑥ draw_fruits( ) 함수 만들기 (2/2)

- subplots( ) 함수 대신에 subplot( ) 함수를 활용해서 만들어도 됨

```
1  def draw_fruits(arr, ratio=1):
2    n = len(arr)
3
4    rows = int(np.ceil(n/10))
5    if rows < 2:
6      cols = n
7    else:
8      cols = 10
9
10   plt.figure(figsize=(cols * ratio, rows * ratio))
11   for j in range(rows):
12     for k in range(cols):
13       if j * 10 + k < n:
14         plt.subplot(rows, cols, j * 10 + k + 1)
15         plt.imshow(arr[j * 10 + k], cmap="gray_r")
16         plt.axis("off")
17
18   plt.show()
```

- ✓ if 조건문이 없다면, 전달되는 arr 배열 내 저장된 이미지의 개수 n이 10의 배수가 아니라면, 이중 for 문에서 IndexError가 발생하게 됨
- ✓ 따라서 if 조건문을 통해서 n개의 이미지에 대해서만 그림을 그릴 수 있도록 함

02. KMeans 클래스

❖ ⑦ 과일 사진 그리기 (1/4)

- draw_fruits( ) 함수를 사용해 레이블이 0인 과일 사진을 모두 그려보자
- km.labels_==0과 같이 쓰면 km.labels_ 배열에서 값이 0인 위치는 True, 그 외는 모두 False가 됨
- 넘파이는 이런 불리언(Boolean) 배열을 사용해 원소를 선택할 수 있음
- 이를 불리언 인덱싱(Boolean Indexing)이라고 함
- 넘파이 배열에 불리언 인덱싱을 적용하면 True인 위치의 원소만 모두 추출함

```
1 fruits[km.labels_==0]
```

실행결과

```
array([[[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
       … (중략) …
       [[0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        ...,
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0],
        [0, 0, 0, ..., 0, 0, 0]]])
```

02. KMeans 클래스

❖ ⑦ 과일 사진 그리기 (2/4)

- 레이블 0으로 클러스터링된 490개의 이미지를 모두 출력해 보자

```
1 draw_fruits(fruits[km.labels_==0])
```

실행결과

이 클러스터는 모두 바나나가 올바르게 모였음

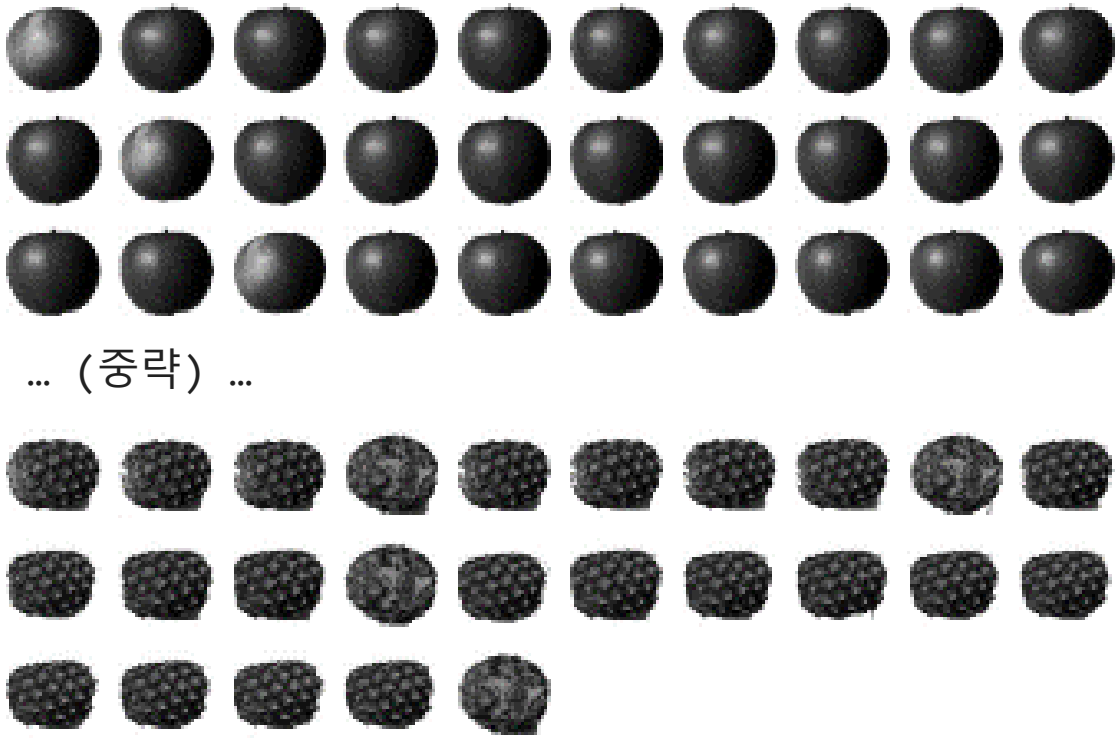
02. KMeans 클래스

❖ ⑦ 과일 사진 그리기 (3/4)

- 레이블 1로 클러스터링된 735개의 이미지를 모두 출력해 보자

```
1 draw_fruits(fruits[km.labels_==1])
```

실행결과



레이블이 1인 클러스터는
사과와 파인애플이 섞여 있음

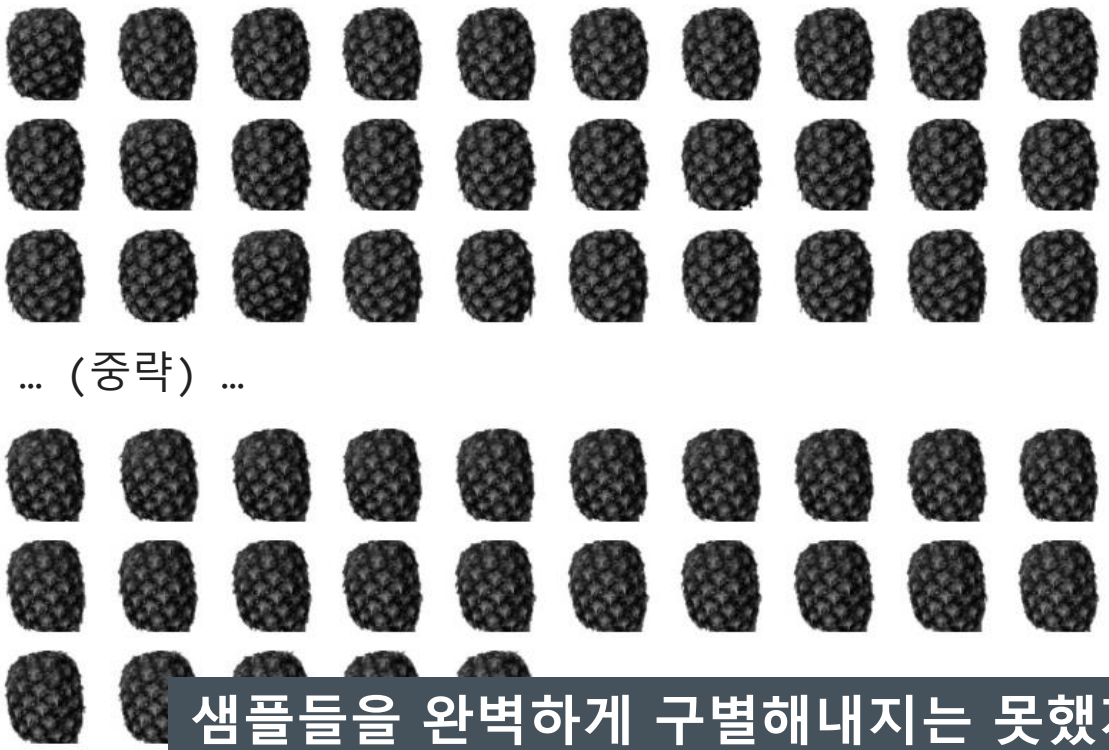
02. KMeans 클래스

❖ ⑦ 과일 사진 그리기 (4/4)

- 레이블 2로 클러스터링된 245개의 이미지를 모두 출력해 보자

```
1 draw_fruits(fruits[km.labels_==2])
```

실행결과



레이블이 2인 클러스터는
파인애플로만 이루어져 있음

샘플들을 완벽하게 구별해내지는 못했지만, 학습 데이터에 레이블을
제공하지 않았음에도 스스로 비슷한 샘플들을 잘 모았음

03. 클러스터 중심

01. k-Means Clustering Algorithm 소개
02. KMeans 클래스
04. 최적의 k 찾기
05. 과일을 자동으로 분류하기
06. 마무리

03. 클러스터 중심

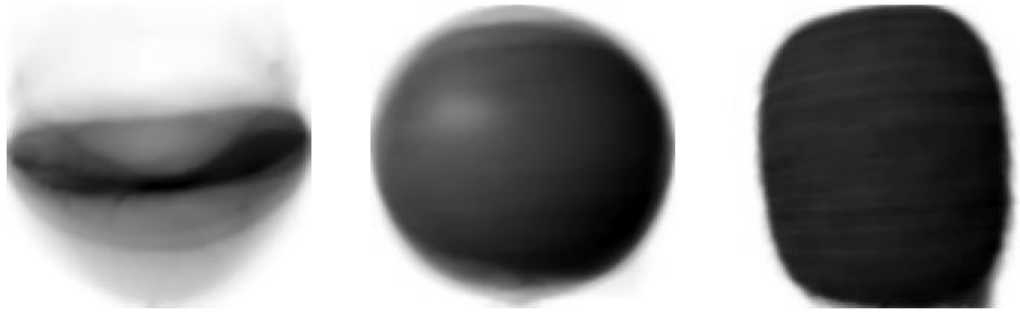
❖ 클러스터 중심 확인하기

- KMeans 클래스가 최종적으로 찾은 클러스터 중심은 cluster_centers_ 속성에 저장되어 있음
- 이 배열은 fruits_2d 샘플의 클러스터 중심이기 때문에
  이미지로 출력하려면 100×100 크기의 2차원 배열로 바꿔야 함

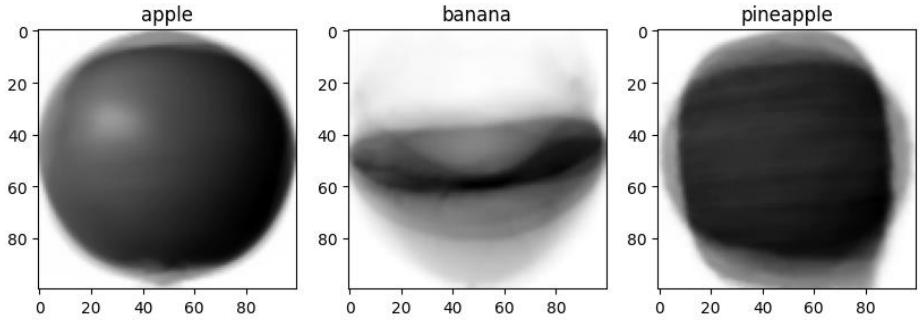
```
1 print(km.cluster_centers_.shape)
2 draw_fruits(km.cluster_centers_.reshape(-1, 100, 100), ratio=3)
```

실행결과

```
(3, 10000)
```



각 과일의 픽셀 평균값



이전 수업시간에서 사과, 바나나, 파인애플의 픽셀 평균값을 출력했던 것과 매우 비슷함

03. 클러스터 중심

❖ transform( ) 메서드 (1/2)

- KMeans 클래스는 학습 데이터 샘플에서 클러스터 중심까지 거리로 변환해 주는 transform( ) 메서드를 갖고 있음
- transform( ) 메서드가 있다는 것은 마치 StandardScaler 클래스처럼 특성값을 변환하는 도구로 사용할 수 있다는 의미임
- transform( ) 메서드는 fit( ) 메서드와 마찬가지로 2차원 배열을 기대함
- fruits_2d[0]처럼 쓰면 (10000,) 크기의 배열이 되므로 에러가 발생함
- 슬라이싱(Slicing) 연산자를 사용해서 (1, 10000) 크기의 배열을 전달해 주자

```
1 print(fruits_2d[0].shape)
2 print(fruits_2d[0:1].shape)
```

실행결과

```
(10000,)
(1, 10000)
```

03. 클러스터 중심

❖ transform( ) 메서드 (2/2)

- transform( ) 메서드를 사용해 첫 번째 샘플의 각 클러스터 중심까지의 거리를 확인해 보자

```
1 print(km.transform(fruits_2d[0:1]))
```

실행결과

```
[[11813.39838108     4012.56512913     8280.55337682]]
```

첫 번째 클러스터 (레이블 0) — 두 번째 클러스터 (레이블 1) — 세 번째 클러스터 (레이블 2)

✓ 하나의 샘플을 전달했기 때문에 반환된 배열은 크기가 (1, 클러스터 개수)인 2차원 배열임
✓ 세 번째 클러스터까지의 거리가 가장 작음
✓ 이 샘플은 레이블 1에 속할 것임

03. 클러스터 중심

❖ predict( ) 메서드

- KMeans 클래스는 가장 가까운 클러스터 중심을 예측 클래스로 출력하는 predict( ) 메서드를 제공함

```
1 print(km.predict(fruits_2d[0:1]))
```

실행결과

```
[1]
```

✓ transform( )의 결과에서 짐작할 수 있듯이 레이블 1로 예측했음
✓ 클러스터 중심을 그려 보았을 때 레이블 1은 사과였으므로 이 샘플은 사과일 것임

```
1 draw_fruits(fruits[0:1])
```

실행결과

draw_fruits( ) 함수로 샘플을 그려본 결과, 사과가 맞음

03. 클러스터 중심

❖ 알고리즘의 반복 수행 횟수

- k-Means Clustering Algorithm은 앞서 동작 원리에서 설명한 바와 같이 반복적으로 클러스터 중심을 옮기면서 최적의 클러스터를 찾음
- 알고리즘이 반복한 횟수는 KMeans 클래스의 n_iter_ 속성에 저장됨

```
1 print(km.n_iter_)
```

실행결과

```
5
```

✓ 클러스터 중심을 사용해 데이터셋을 저차원(이 경우에는 10,000에서 3으로 줄임)으로 변환 할 수 있었음
✓ 또는 가장 가까운 거리에 있는 클러스터 중심을 샘플의 예측값으로 사용할 수 있다는 것을 배웠음

03. 클러스터 중심

❖ 클러스터 개수

- 이번에 우리는 타겟값을 사용하지 않았지만 약간의 편법을 사용했음
- n_clusters를 3으로 지정한 것은 타겟에 대한 정보를 활용한 셈임
- 실전에서는 클러스터 개수조차 알 수 없음

그렇다면 n_cluster를 어떻게 지정해야 할까?
최적의 클러스터 개수는 얼마일까?

04. 최적의 k 찾기

04. 최적의 k 찾기

❖ 엘보우 방법(Elbow Method) (1/5)

k-Means Clustering Algorithm의 단점

- ✓ 클러스터 개수를 사전에 지정해야 한다는 것임
- ✓ 실전에서는 몇 개의 클러스터가 있는지 알 수 없음

- ✓ 어떻게 하면 적절한 k 값을 찾을 수 있을까?
- ✓ 사실 군집 알고리즘에서 적절한 k 값을 찾기 위한 방법은 없음
- ✓ 몇 가지 방법이 있지만 저마다 장단점이 있음

이번 수업에서는 적절한 클러스터 개수를 찾기 위한 대표적인 방법인 엘보우 방법에 대해서 알아 보겠음

04. 최적의 k 찾기

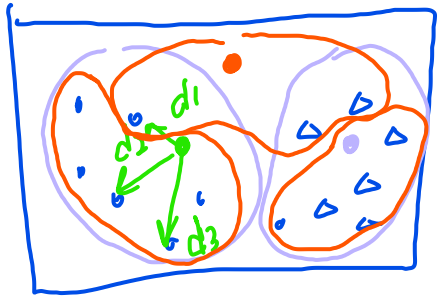
❖ 엘보우 방법(Elbow Method) (2/5)

이너셔(Inertia) $\approx \sum_{i=1}^{N} (d_i)^2$

- ✓ k-Means Clustering Algorithm은 클러스터 중심과 클러스터에 속한 샘플 사이의 거리를 잴 수 있음
- ✓ 이 거리의 제곱 합을 이너셔(Inertia)라고 부름
- ✓ 이너셔는 클러스터에 속한 샘플이 얼마나 가깝게 모여 있는지를 나타내는 값으로 생각할 수 있음

클러스터 개수가 늘어남 ▶ 클러스터 개개의 크기가 줄어듦 ▶ 이너셔가 줄어듦

k↑ N↓

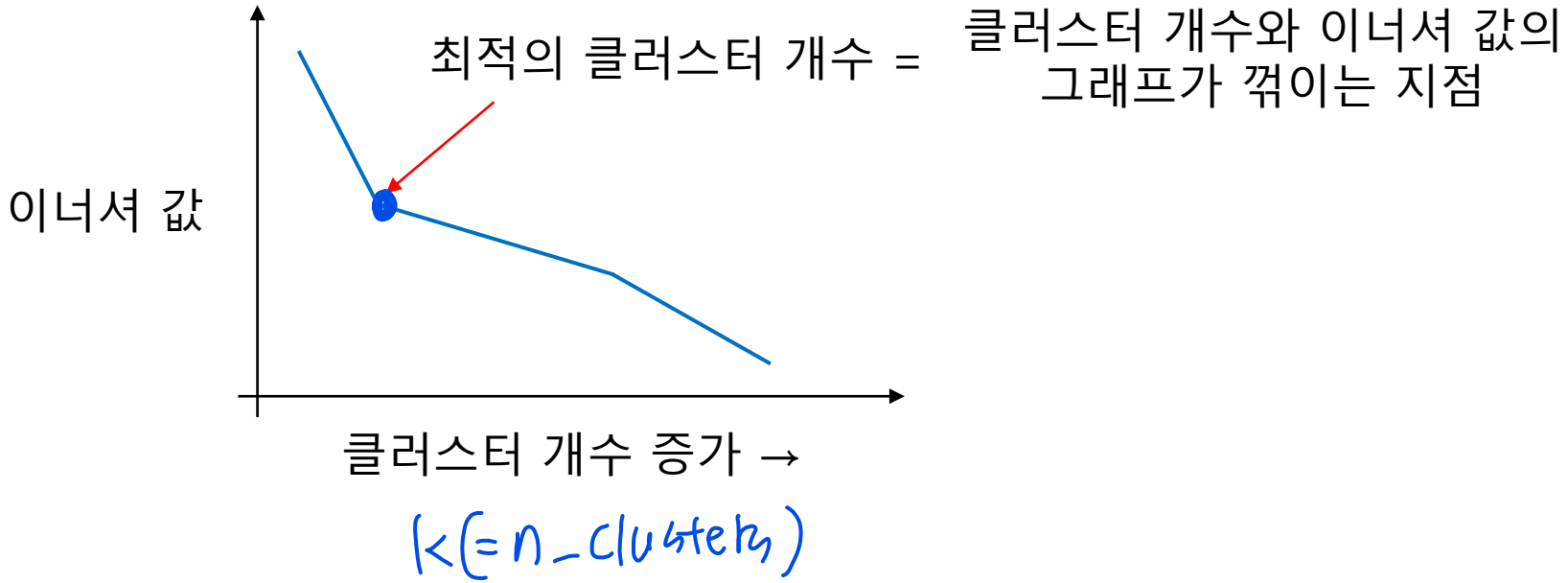


엘보우 방법은 클러스터 개수를 늘려가면서
이너셔의 변화를 관찰하여 최적의 클러스터 개수를 찾는 방법임

04. 최적의 k 찾기

❖ 엘보우 방법(Elbow Method) (3/5)

- 클러스터 개수를 증가시키면서 이너셔를 그래프로 그리면 감소하는 속도가 꺾이는 지점이 있음
- 이 지점부터는 클러스터 개수를 늘려도 클러스터에 잘 밀집된 정도가 크게 개선되지 않음
- 즉, 이너셔가 크게 줄어들지 않음
- 이 지점이 마치 팔꿈치 모양이어서 엘보우 방법이라 부름



04. 최적의 k 찾기

❖ 엘보우 방법(Elbow Method) (4/5)

- KMeans 클래스는 자동으로 이너셔를 계산해서 inertia_ 속성으로 제공함

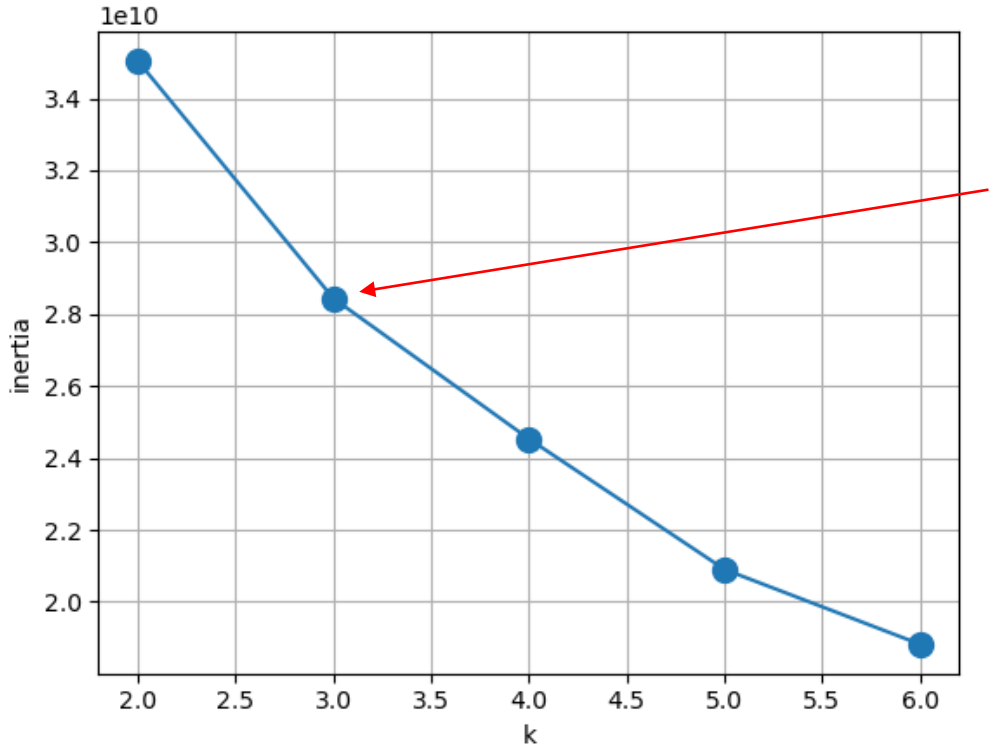
```
inertia = []
for k in range(2, 7):
  km = KMeans(n_clusters=k, init="random", random_state=42)
  km.fit(fruits_2d)
  inertia.append(km.inertia_)

plt.figure()
plt.plot(range(2, 7), inertia, marker='o', markersize=10)
plt.xlabel('k')
plt.ylabel("inertia")
plt.grid(True)
plt.show()
```

✓ 클러스터 개수 k를 2~6까지 바꿔가며 KMeans 클래스를 5번 훈련함
✓ fit( ) 메서드로 모델을 훈련한 후 inertia_ 속성에 저장된 이너셔 값을 inertia 리스트에 추가함
✓ 마지막으로 inertia 리스트에 저장된 값을 그래프로 출력함

04. 최적의 k 찾기

❖ 엘보우 방법(Elbow Method) (5/5)

실행결과



- ✓ 이 그래프에서 꺾이는 지점이 뚜렷하지는 않지만, k=3에서 그래프의 기울기가 조금 바뀐 것을 볼 수 있음
- ✓ 엘보우 지점보다 클러스터 개수가 많아지면 이너셔의 변화가 줄어들면서 군집 효과도 줄어듦

05. 과일을 자동으로 분류하기

05. 과일을 자동으로 분류하기

❖ 전체 과정 요약 (1/3)

- 이전 수업에서는 과일 종류별로 픽셀 평균값을 계산했음
- 하지만 실전에서는 어떤 과일 사진이 주어질지 모름
- 따라서 타겟값을 모르는 상태에서 자동으로 사진을 클러스터로 모을 수 있는 군집 알고리즘이 필요함
- 이번 수업에서는 대표적인 군집 알고리즘인 k-Means Clustering Algorithm을 살펴보았음
- k-Means Clustering Algorithm은 비교적 간단하고 속도가 빠르며 이해하기도 쉬움
- k-Means Clustering Algorithm을 구현한 scikit-learn의 KMeans 클래스는
  각 샘플이 어떤 클래스에 소속되어 있는지 labels_ 속성에 저장함

05. 과일을 자동으로 분류하기

❖ 전체 과정 요약 (2/3)

- 각 샘플에서 각 클러스터까지의 거리를 하나의 특성으로 활용할 수도 있음
- 이를 위해 KMeans 클래스는 transform( ) 메서드를 제공함
- 또한 predict( ) 메서드에서 새로운 샘플에 대해 가장 가까운 클러스터를 예측값으로 출력함
- k-Means Clustering Algorithm은 사전에 클러스터 개수를 미리 지정해야 함
- 사실 데이터를 직접 확인하지 않고서는 몇 개의 클러스터가 만들어질지 알기 어려움
- 최적의 클러스터 개수 k를 알아내는 한 가지 방법은
  클러스터가 얼마나 밀집되어 있는지 나타내는 이너셔(Inertia)를 사용하는 것임
- 이너셔가 더 이상 줄어들지 않는다면 클러스터 개수를 더 늘리는 것은 효과가 없음
- 이를 엘보우 방법(Elbow Method)라고 부름

05. 과일을 자동으로 분류하기

❖ 전체 과정 요약 (3/3)

- scikit-learn의 KMeans 클래스는 자동으로 이너셔를 계산하여 inertia_ 속성으로 제공함
- 클러스터 개수를 늘리면서 반복하여 KMeans 알고리즘을 학습시키고 이너셔가 줄어드는 속도가 꺾이는 지점을 최적의 클러스터 개수로 결정함
- 이번 수업에서는 k-Means Clustering Algorithm의 클러스터 중심까지 거리를 특성으로 사용할 수도 있다는 점을 보았음
- 이렇게 하면 학습 데이터의 차원을 크게 줄일 수 있음
- 데이터셋의 차원을 줄이면 지도 학습 알고리즘의 속도를 크게 높일 수 있음

다음 수업 시간에는 비지도 학습의 또 다른 종류인 차원 축소에 대해서 알아보겠음

05. 과일을 자동으로 분류하기

❖ 전체 소스 코드 (1/3)

```
# 02. KMeans 클래스
import os
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
cur_dir = os.getcwd()
fruit_list = [“Apple”, “Banana”, “Pineapple”]
fruit_npy = []
for fruit_name in fruit_list:
  folder_name = cur_dir + “\\” + fruit_name
  file_list = os.listdir(folder_name)
  for file_name in file_list:
    img = mpimg.imread(folder_name + “\\” + file_name)
    R, G, B = img[:,:,0], img[:,:,1], img[:,:,2]
    imgGray = 0.299 * R + 0.587 * G + 0.114 * B
    imgGray = np.array(imgGray, dtype=‘int’)
    imgGray2 = 255 – imgGray
    fruit_npy.append(imgGray2)
fruit_npy = np.array(fruit_npy)
print(fruit_npy.shape)
```

```
np.save(“fruits.npy”, fruit_npy)

fruits = np.load(“fruits.npy”)

print(fruits.shape)

fruits_2d = fruits.reshape(-1, 100*100)
print(fruits_2d.shape)

from sklearn.cluster import KMeans

km = KMeans(n_clusters=3, init=“random”,
            random_state=42)
km.fit(fruits_2d)

print(km.labels_)
print(km.labels_.shape)

print(np.unique(km.labels_, return_counts=True))

```

05. 과일을 자동으로 분류하기

❖ 전체 소스 코드 (2/3)

```
41  def draw_fruits(arr, ratio=1):
42    n = len(arr)
43
44    rows = int(np.ceil(n/10))
45    cols = n if rows < 2 else 10
46
47    fig, axs = plt.subplots(rows, cols,
48                    figsize=(cols * ratio, rows * ratio),
49                    squeeze=False)
50
51    for j in range(rows):
52      for k in range(cols):
53        if j * 10 + k < n:
54          axs[j, k].imshow(arr[j * 10 + k],
55                           cmap="gray_r")
56
57        axs[j, k].axis("off")
58
59    plt.show()
60
```

```
61  fruits[km.labels_==0]
62
63  draw_fruits(fruits[km.labels_==0])
64  draw_fruits(fruits[km.labels_==1])
65  draw_fruits(fruits[km.labels_==2])
66
67  # 03. 클러스터 중심
68  print(km.cluster_centers_.shape)
69  draw_fruits(km.cluster_centers_.reshape(-1, 100, 100),
70              ratio=3)
71
72  print(fruits_2d[0].shape)
73  print(fruits_2d[0:1].shape)
74
75  print(km.transform(fruits_2d[0:1]))
76
77  print(km.predict(fruits_2d[0:1]))
78  draw_fruits(fruits[0:1])
79
80  print(km.n_iter_)
```

05. 과일을 자동으로 분류하기

❖ 전체 소스 코드 (3/3)

```
81  # 04. 최적의 k 찾기
82  inertia = []
83  for k in range(2, 7):
84    km = KMeans(n_clusters=k, init="random",
85                random_state=42)
86    km.fit(fruits_2d)
87    inertia.append(km.inertia_)
88
89  plt.figure()
90  plt.plot(range(2, 7), inertia, marker='o',
91           markersize=10)
92  plt.xlabel('k')
93  plt.ylabel("inertia")
94  plt.grid(True)
95  plt.show()
```

06. 마무리

01. k-Means Clustering Algorithm 소개
02. KMeans 클래스
03. 클러스터 중심
04. 최적의 k 찾기
05. 과일을 자동으로 분류하기

06. 마무리

❖ 키워드로 끝내는 핵심 포인트 (1/2)

k-Means Clustering Algorithm

- ✓ 처음에 랜덤하게 클러스터 중심을 정하고 클러스터를 만듦
- ✓ 그 다음 클러스터의 중심을 이동하고 다시 클러스터를 만드는 식으로 반복해서 최적의 클러스터를 구성하는 알고리즘임

클러스터 중심(Cluster Center, Centroid)

- ✓ k-평균 알고리즘이 만든 클러스터에 속한 샘플의 특성 평균값임
- ✓ 가장 가까운 클러스터 중심을 샘플의 또 다른 특성으로 사용하거나 새로운 샘플에 대한 예측으로 활용할 수 있음

06. 마무리

❖ 키워드로 끝내는 핵심 포인트 (2/2)

엘보우 방법(Elbow Method)

- ✓ 최적의 클러스터 개수를 정하는 방법 중 하나임
- ✓ 이너셔(Inertia)는 클러스터 중심과 샘플 사이 거리의 제곱 합임
- ✓ 클러스터 개수에 따라 이너셔가 꺾이는 지점이 적절한 클러스터 개수 k가 될 수 있음
- ✓ 이 그래프의 모양을 따서 엘보우 방법이라고 부름

06. 마무리

❖ 핵심 패키지와 함수: scikit-learn

KMeans

- ✓ k-Means Clustering Algorithm 클래스임
- ✓ n_cluster에는 클러스터 개수를 지정하며, 기본값은 8임
- ✓ 처음에는 랜덤하게 센트로이드를 초기화하기 때문에 여러 번 반복하여 이너셔를 기준으로 가장 좋은 결과를 선택함
- ✓ n_init는 이 반복 횟수를 지정하며, 기본값은 10임
- ✓ max_iter는 k-Means Clustering Algorithm의 한 번 실행에서 최적의 센트로이드를 찾기 위해 반복할 수 있는 최대 횟수이며, 기본값은 200임

06. 마무리

❖ 확인 문제 1.

k-Means Clustering 알고리즘에서 클러스터를 표현하는 방법이 아닌 것은 무엇인가요?

① 클러스터에 속한 샘플의 평균
② 클러스터 중심
③ 센트로이드
④ 클러스터에 속한 샘플 개수

06. 마무리

❖ 확인 문제 2.

k-Means Clustering 알고리즘에서 최적의 클러스터 개수는 어떻게 정할 수 있나요?

① 엘보우 방법을 사용해 이너셔의 감소 정도가 꺾이는 클러스터 개수를 찾습니다.

② 랜덤하게 클러스터 개수를 정해서 k-Means Clustering 알고리즘을 훈련하고 가장 낮은 이너셔가 나오는 클러스터 개수를 찾습니다.

③ 학습 데이터셋을 모두 조사하여 몇 개의 클러스터가 나올 수 있는지 직접 확인합니다.

④ 교차 검증을 사용하여 최적의 클러스터 개수를 찾습니다.

지도학습

끝맺음

- ❖ 01. k-Means Clustering Algorithm 소개
- ❖ 02. KMeans 클래스
- ❖ 03. 클러스터 중심
- ❖ 04. 최적의 k 찾기
- ❖ 05. 과일을 자동으로 분류하기
- ❖ 06. 마무리

THANK YOU!
Q & A

- Name: 권범
- Office: 동덕여자대학교 인문관 B821호
- Phone: 02-940-4752
- E-mail: bkwon@dongduk.ac.kr