CHAPTER 06 비지도 학습

인공지능

대전대학교 컴퓨터공학과

조교수 박상돈

학습 로드맵

머신러닝편 **01~06장**

딥러닝만 먼저 배우고
싶다면 01~04장을 읽은 후
07장으로 건너뛰어도 좋습니다.

딥러닝편 **07~10장**

07장을 읽은 후 08장과 09장은
순서대로 읽지 않아도 괜찮습니다.
10장을 읽기 전에 07장과 09장을
읽는 것이 좋습니다.

난이도 ●●●●●

START

01 나의 첫 머신러닝 ●○○○○

02 데이터 다루기 ●●○○○

03 회귀 알고리즘과 모델 규제

2번 보기

04 다양한 분류 알고리즘 ●●○○○

05 트리 알고리즘 ●●●○○

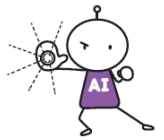
06 비지도 학습 ●●●○○

07 딥러닝을 시작합니다 ●●●●○

08 이미지를 위한 인공 신경망 ●●●●●

09 텍스트를 위한 인공 신경망 ●●●●●

10 언어 모델을 위한 신경망 ●●●●●



GOAL

Contents

CHAPTER 06 비지도 학습

SECTION 6-1 군집 알고리즘

SECTION 6-2 k-평균

SECTION 6-3 주성분 분석

CHAPTER 06 비지도 학습

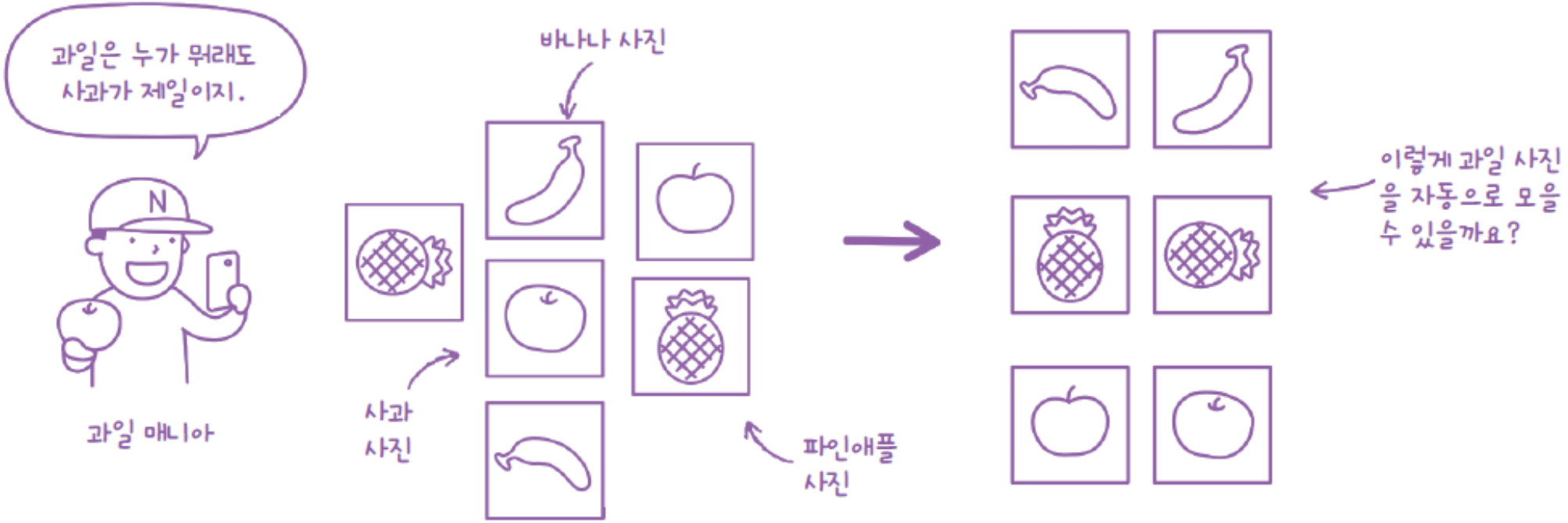
비슷한 과일끼리 모으자!

학습목표

- 타깃이 없는 데이터를 사용하는 비지도 학습과 대표적인 알고리즘을 소개합니다.
- 대표적인 군집 알고리즘인 k-평균과 DBSCAN을 배웁니다.
- 대표적인 차원 축소 알고리즘인 주성분 분석(PCA)을 배웁니다.

SECTION 6-1 군집 알고리즘(1)

- 타깃을 모르는 비지도 학습
  - 타깃을 모르는 사진을 종류별로 분류
  - 타깃이 없을 때 사용하는 머신러닝 알고리즘이 비지도 학습(unsupervised learning)



SECTION 6-1 군집 알고리즘(2)

◦ 과일 사진 데이터 준비하기

- 준비한 과일 데이터는 사과, 바나나, 파인애플을 담고 있는 흑백 사진
- 이 데이터는 넘파이 배열의 기본 저장 포맷인 npy 파일로 저장되어 있음
- 코랩에서 다음 명령을 실행해 파일을 다운로드

```
!wget https://bit.ly/fruits_300_data -0 fruits_300.npy
```

- 넘파이와 맷플롯립 패키지를 임포트

```
import numpy as np
import matplotlib.pyplot as plt
```

- 넘파이에서 load( ) 메서드에 파일 이름을 전달하여 npy 파일을 로드

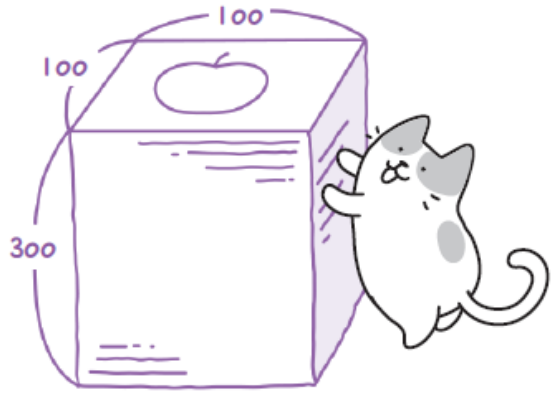
```
fruits = np.load('fruits_300.npy')
```

- fruits 배열의 크기를 확인

```
print(fruits.shape)
```

→ (300, 100, 100)

(샘플갯수, 이미지 높이, 이미지 너비)



SECTION 6-1 군집 알고리즘(3)

◦ 과일 사진 데이터 준비하기

- 첫 번째 이미지의 첫 번째 행에 들어 있는 픽셀 100개 값을 출력
  흑백 사진을 담고 있으므로 0~255의 정숫값을 가짐

```
print(fruits[0, 0, :])
```

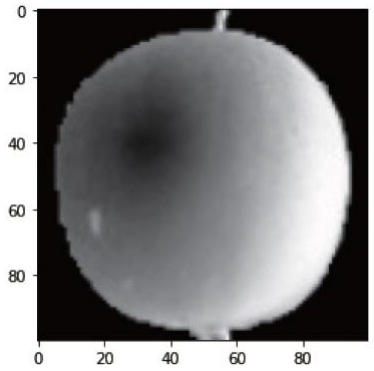
→

```
[  1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   2   1
   2   2   2   2   2   2   1   1   1   1   1   1   1   1   2   3   2   1
   2   1   1   1   1   2   1   3   2   1   3   1   4   1   2   5   5   5
  19 148 192 117  28   1   1   2   1   4   1   1   3   1   1   1   1   1
   2   2   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1   1
   1   1   1   1   1   1   1   1   1   1]
```

- 맷플롯립의 imshow( ) 함수를 사용하여 넘파이 배열로 저장된 이미지를 구현
  흑백 이미지이므로 cmap 매개변수를 'gray'로 지정

```
plt.imshow(fruits[0], cmap='gray')
plt.show()
```

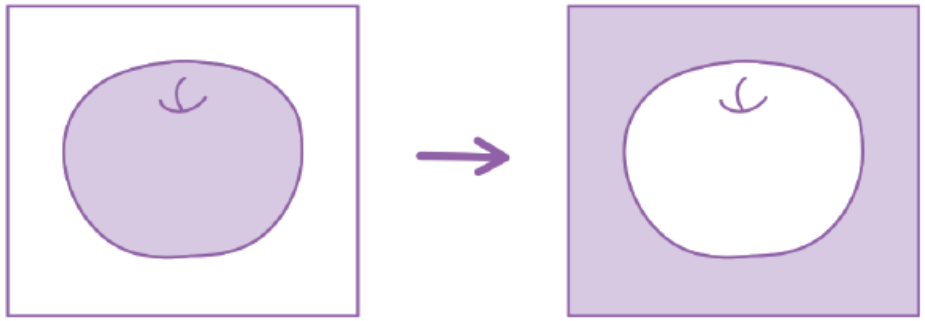
→



SECTION 6-1 군집 알고리즘(4)

◦ 과일 사진 데이터 준비하기

- 흑백 이미지는 사진으로 찍은 이미지를 넘파이 배열로 변환할 때 반전시킨 것
- 사진의 흰 바탕(높은 값)은 검은색(낮은 값)으로 만들고 실제 사과가 있어 짙은 부분(낮은 값)은 밝은색(높은 값)으로 변환
- 흰색 바탕은 우리에게 중요하지 않지만 컴퓨터는 255에 가까운 바탕에 집중. 따라서 바탕을 검게 만들고 사진에 짙게 나온 사과를 밝은색으로 만듦



✚ **여기서 잠깐 컴퓨터는 왜 255에 가까운 바탕에 집중하나?**

- 알고리즘이 어떤 출력을 만들기 위해 곱셈, 덧셈을 수행
- 픽셀값이 0이면 출력도 0이 되어 의미가 없음
- 픽셀값이 높으면 출력값도 커지기 때문에 의미를 부여하기 좋음

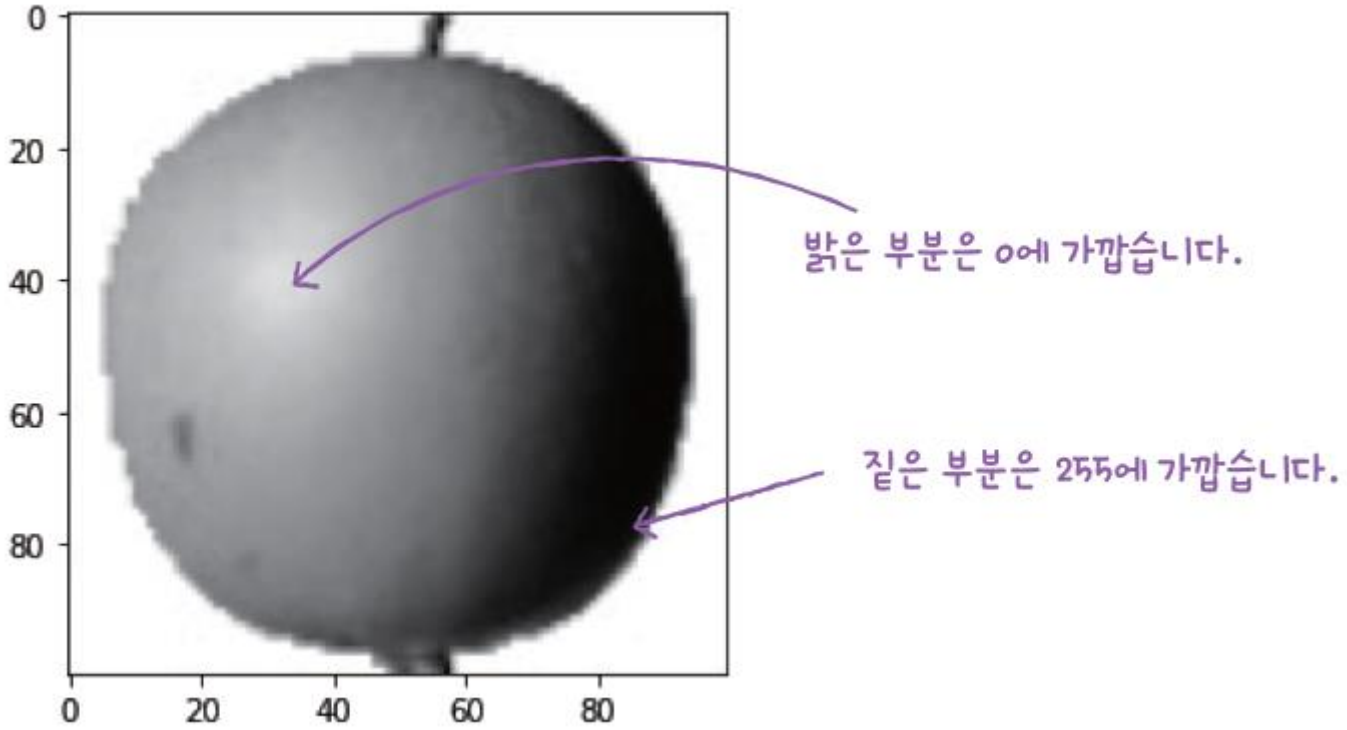
SECTION 6-1 군집 알고리즘(5)

◦ 과일 사진 데이터 준비하기

- cmap 매개변수를 'gray_r'로 지정하면 다시 반전하여 우리 눈에 보기 좋게 출력

```
plt.imshow(fruits[0], cmap='gray_r')
plt.show()
```

→

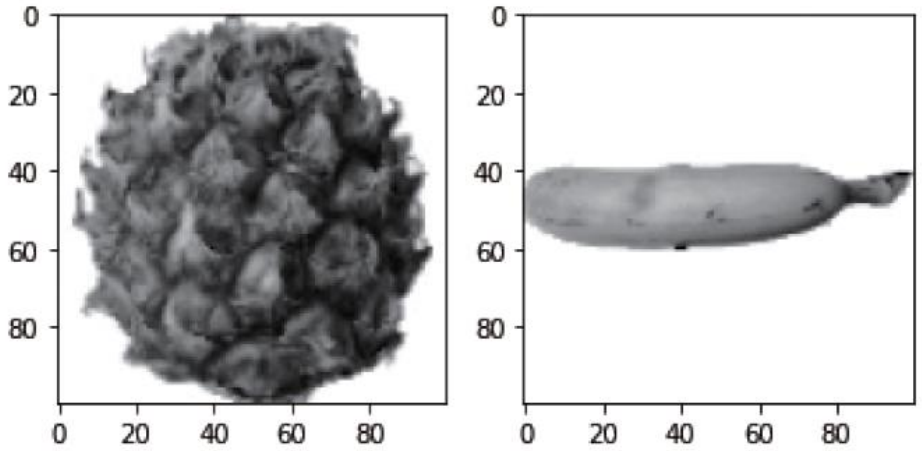


SECTION 6-1 군집 알고리즘(6)

◦ 과일 사진 데이터 준비하기

- 바나나와 파인애플 이미지도 출력

```
fig, axs = plt.subplots(1, 2)
axs[0].imshow(fruits[100], cmap='gray_r')
axs[1].imshow(fruits[200], cmap='gray_r')
plt.show()
```

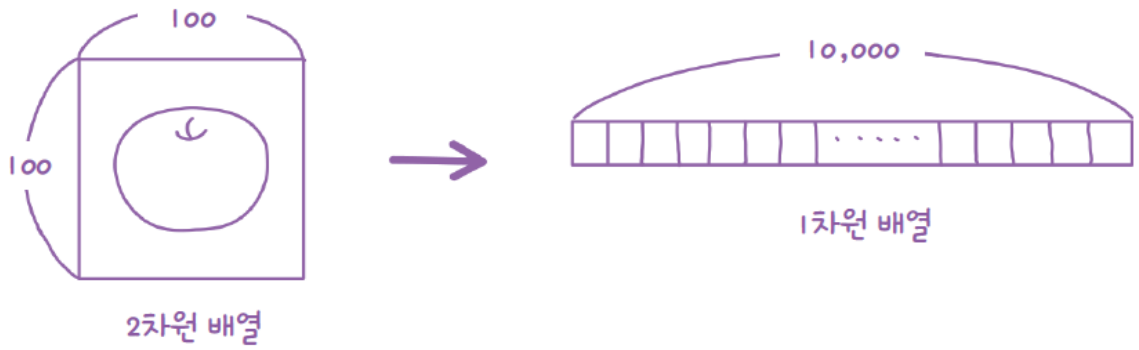


- 맷플롯립의 subplots( ) 함수를 사용하면 여러 개의 그래프를 배열처럼 쌓을 수 있도록 도와줌
- subplots( ) 함수의 두 매개변수는 그래프를 쌓을 행과 열을 지정
- 위에서는 subplots(1, 2)처럼 하나의 행과 2개의 열을 지정

SECTION 6-1 군집 알고리즘(7)

◦ 픽셀값 분석하기

- 사용하기 쉽게 fruits 데이터를 사과, 파인애플, 바나나로 각각 나누기
- 넘파이 배열을 나눌 때 100 × 100 이미지를 펼쳐서 길이가 10,000인 1차원 배열로 만들기
  - 이렇게 펼치면 이미지로 출력하긴 어렵지만 배열을 계산할 때 편리



- fruits 배열에서 순서대로 100개씩 선택하기 위해 슬라이싱 연산자를 사용. 그다음 reshape( ) 메서드를 사용해 두 번째 차원(100)과 세 번째 차원(100)을 10,000으로 합침

```
apple = fruits[0:100].reshape(-1, 100*100)
pineapple = fruits[100:200].reshape(-1, 100*100)
banana = fruits[200:300].reshape(-1, 100*100)
```

- 이제 apple, pineapple, banana 배열의 크기는 (100, 10000)이 됨

SECTION 6-1 군집 알고리즘(8)

- 픽셀값 분석하기
  - apple 배열의 mean( ) 메서드로 각 샘플의 픽셀 평균값을 계산

```
print(apple.mean(axis=1))
```

→

```
[ 88.3346  97.9249  87.3709  98.3703  92.8705  82.6439  94.4244  95.5999
  90.681   81.6226  87.0578  95.0745  93.8416  87.017   97.5078  87.2019
  88.9827 100.9158  92.7823 100.9184 104.9854  88.674   99.5643  97.2495
  94.1179  92.1935  95.1671  93.3322 102.8967  94.6695  90.5285  89.0744
  97.7641  97.2938 100.7564  90.5236 100.2542  85.8452  96.4615  97.1492
  90.711  102.3193  87.1629  89.8751  86.7327  86.3991  95.2865  89.1709
  96.8163  91.6604  96.1065  99.6829  94.9718  87.4812  89.2596  89.5268
  93.799   97.3983  87.151   97.825  103.22    94.4239  83.6657  83.5159
 102.8453  87.0379  91.2742 100.4848  93.8388  90.8568  97.4616  97.5022
  82.446   87.1789  96.9206  90.3135  90.565   97.6538  98.0919  93.6252
  87.3867  84.7073  89.1135  86.7646  88.7301  86.643   96.7323  97.2604
  81.9424  87.1687  97.2066  83.4712  95.9781  91.8096  98.4086 100.7823
 101.556  100.7027  91.6098  88.8976]
```

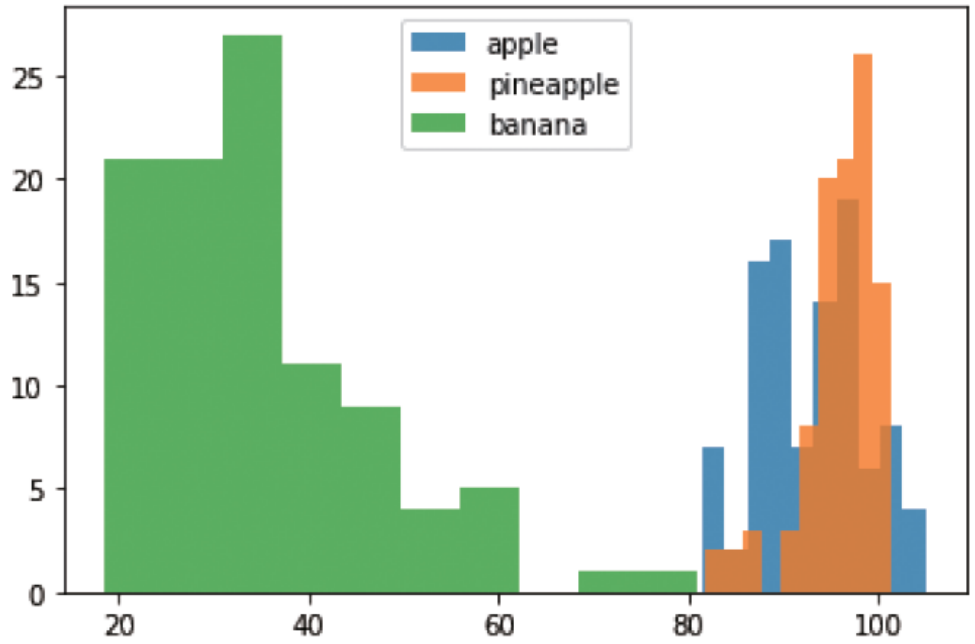
SECTION 6-1 군집 알고리즘(9)

◦ 픽셀값 분석하기

- 맷플롯립의 hist( ) 함수로 평균값 분포 히스토그램을 구현
  - legend( ) 함수를 사용해 어떤 과일의 히스토그램인지 범례 추가

```
plt.hist(np.mean(apple, axis=1), alpha=0.8)
plt.hist(np.mean(pineapple, axis=1), alpha=0.8)
plt.hist(np.mean(banana, axis=1), alpha=0.8)
plt.legend(['apple', 'pineapple', 'banana'])
plt.show()
```

→



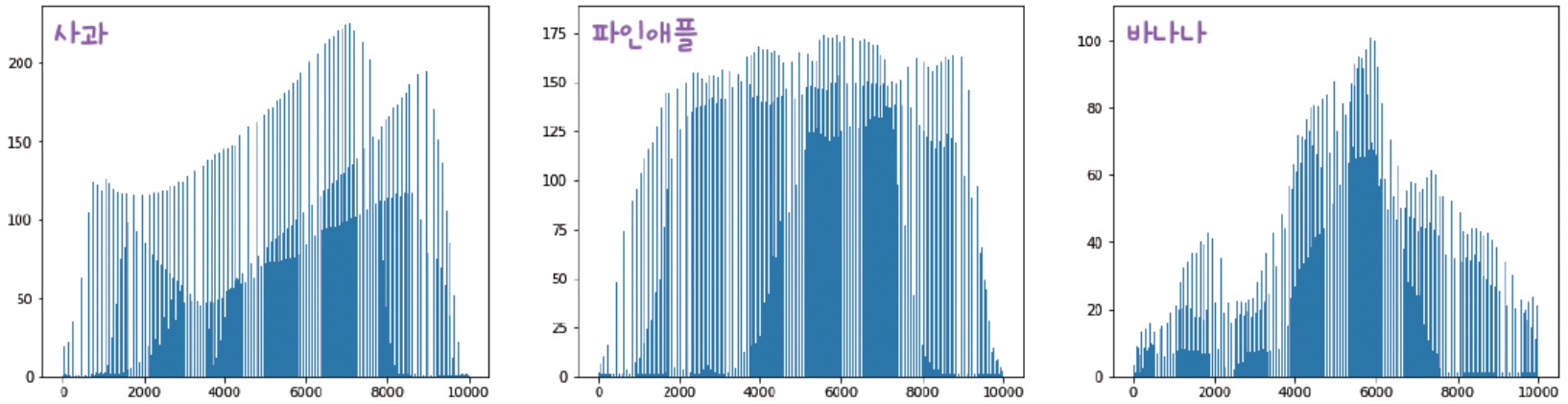
- 바나나는 픽셀 평균값만으로 사과나 파인애플과 확실히 구분됨
  - 바나나는 사진에서 차지하는 영역이 작기 때문에 평균값이 작음
- 반면 사과와 파인애플은 많이 겹쳐있어서 픽셀값만으로는 구분하기 쉽지 않음
  - 사과나 파인애플은 대체로 형태가 동그랗고 사진에서 차지하는 크기도 비슷하기 때문

SECTION 6-1 군집 알고리즘(10)

◦ 픽셀값 분석하기

- 샘플의 평균값이 아니라 픽셀별 평균값 비교
  - 픽셀의 평균을 계산하기 위해 axis=0으로 지정
- 맷플롯립의
- bar( ) 함수를 사용해 픽셀 10,000개에 대한 평균값을 막대그래프로 구현

```
fig, axs = plt.subplots(1, 3, figsize=(20,5))
axs[0].bar(range(10000), np.mean(apple, axis=0))
axs[1].bar(range(10000), np.mean(pineapple, axis=0))
axs[2].bar(range(10000), np.mean(banana, axis=0))
plt.show()
```

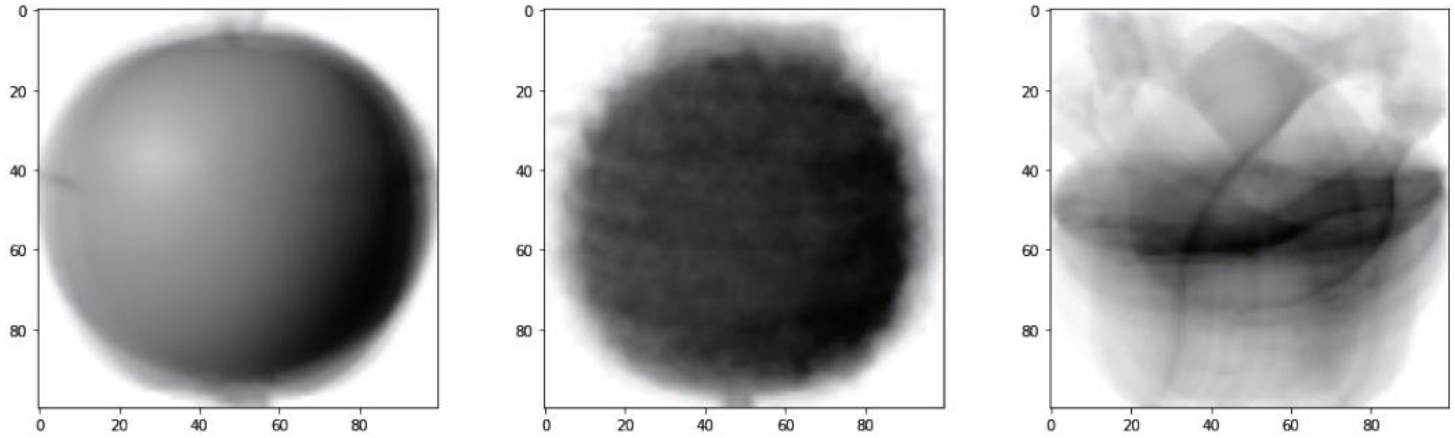


SECTION 6-1 군집 알고리즘(11)

- 픽셀값 분석하기
  - 픽셀 평균값을 100 × 100 크기로 바꿔서 이미지처럼 출력하여 앞의 그래프와 비교

```
apple_mean = np.mean(apple, axis=0).reshape(100, 100)
pineapple_mean = np.mean(pineapple, axis=0).reshape(100, 100)
banana_mean = np.mean(banana, axis=0).reshape(100, 100)
fig, axs = plt.subplots(1, 3, figsize=(20,5))
axs[0].imshow(apple_mean, cmap='gray_r')
axs[1].imshow(pineapple_mean, cmap='gray_r')
axs[2].imshow(banana_mean, cmap='gray_r')
plt.show()
```

→



SECTION 6-1 군집 알고리즘(12)

- 평균값과 가까운 사진 고르기
  - 사과 사진의 평균값인 apple_mean과 가장 가까운 사진 고르기
    -3장에서 학습한 절댓값 오차를 사용
  - fruits 배열에 있는 모든 샘플에서 apple_mean을 뺀 절댓값의 평균을 계산

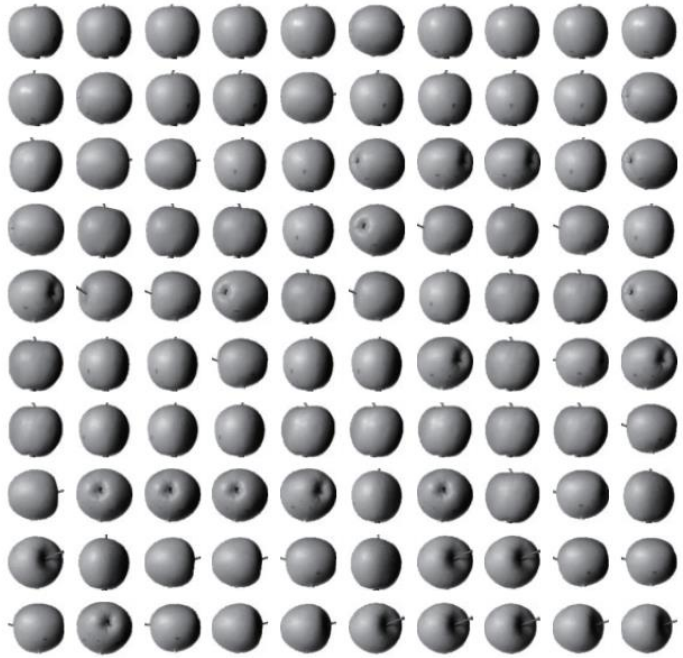
```
abs_diff = np.abs(fruits - apple_mean)
abs_mean = np.mean(abs_diff, axis=(1,2))
print(abs_mean.shape)
```

→ (300,)

  - apple_mean과 오차가 가장 작은 샘플 100개 고르기

```
apple_index = np.argsort(abs_mean)[:100]
fig, axs = plt.subplots(10, 10, figsize=(10,10))
for i in range(10):
    for j in range(10):
        axs[i, j].imshow(fruits[apple_index[i*10 + j]], cmap='gray_r')
        axs[i, j].axis('off')
plt.show()
```

→



  - 군집(clustering): 비슷한 샘플끼리 그룹으로 모으는 작업
  - 클러스터(cluster): 군집 알고리즘에서 만든 그룹

SECTION 6-1 군집 알고리즘(13)

- 비슷한 샘플끼리 모으기(문제해결 과정)
  - 문제
    - 고객들이 올린 과일 사진을 자동으로 모으기
    - 어떤 과일 사진을 올릴지 미리 예상할 수 없기 때문에 타깃값을 준비하여 분류 모델을 훈련하기 어려움
  - 해결
    - 비지도 학습: 타깃값이 없을 때 데이터에 있는 패턴을 찾거나 데이터 구조를 파악하는 머신러닝 방식
    - 타깃이 없기 때문에 알고리즘을 직접적으로 가르칠 수가 없고, 대신 알고리즘은 스스로 데이터가 어떻게 구성되어 있는지 분석
    - 대표적인 비지도 학습 문제는 '군집'
      - 군집은 비슷한 샘플끼리 그룹으로 모으는 작업
      - 이 절에서는 사진의 픽셀을 사용해 군집과 비슷한 작업을 수행, 샘플이 어떤 과일인지 미리 알고 있었기 때문에 사과 사진의 평균값을 알 수 있었음
    - 실제 비지도 학습에서는 타깃이 없는 사진을 사용

SECTION 6-1 마무리

- 키워드로 끝나는 핵심 포인트
  - 비지도 학습은 머신러닝의 한 종류로 훈련 데이터에 타깃이 없음
    - 타깃이 없기 때문에 외부의 도움 없이 스스로 유용한 무언가를 학습해야 함
    - 대표적인 비지도 학습 작업은 군집, 차원 축소 등
  - 히스토그램은 구간별로 값이 발생한 빈도를 그래프로 표시한 것
    - 보통 x축이 값의 구간(계급)이고, y축은 발생 빈도(도수)
  - 군집은 비슷한 샘플끼리 하나의 그룹으로 모으는 대표적인 비지도 학습 작업
    - 클러스터: 군집 알고리즘으로 모은 샘플 그룹

SECTION 6-1 확인 문제

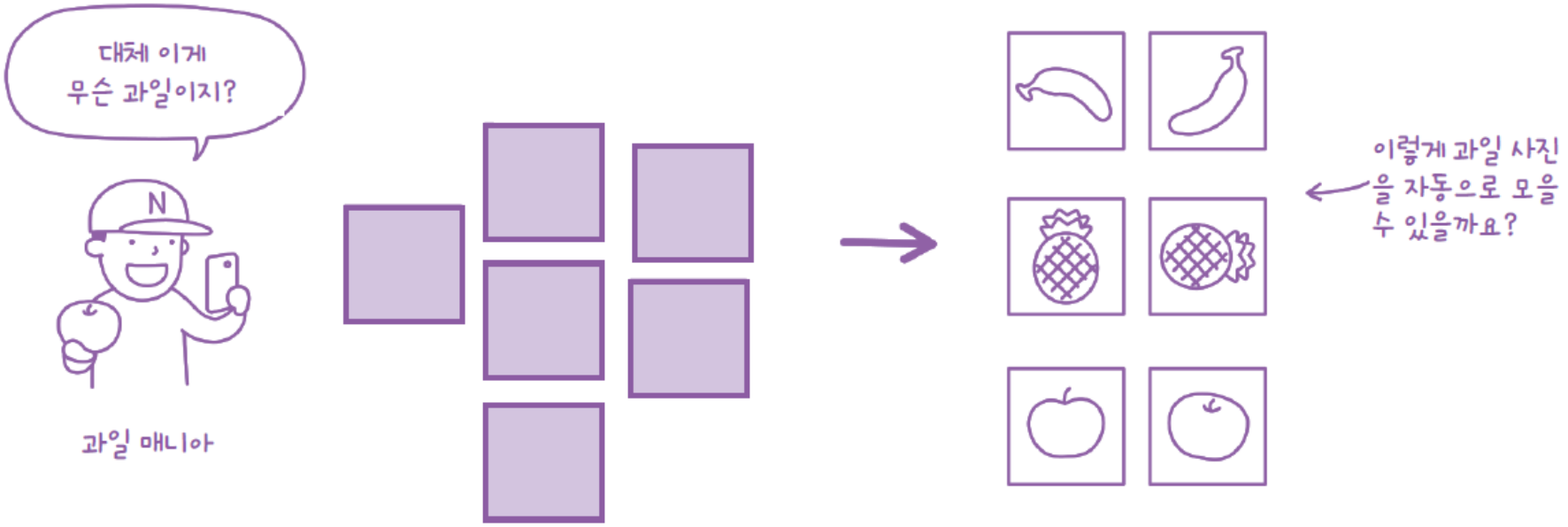
1. 히스토그램을 그릴 수 있는 맷플롯립 함수는?

   ① hist( )
   ② scatter( )
   ③ plot( )
   ④ bar( )

2. 본문에서 했던 것처럼 바나나 사진의 평균 banana_mean과 비슷한 사진 100장을 찾아 출력. 바나나 사진을 모두 찾을 수 있나?

SECTION 6-2 k-평균(1)

- k-평균(k-means) 군집 알고리즘이 평균값을 자동으로 찾아줌
- 이 평균값이 클러스터의 중심에 위치하기 때문에 클러스터 중심(cluster center) 또는 센트로이드(centroid)라고 함

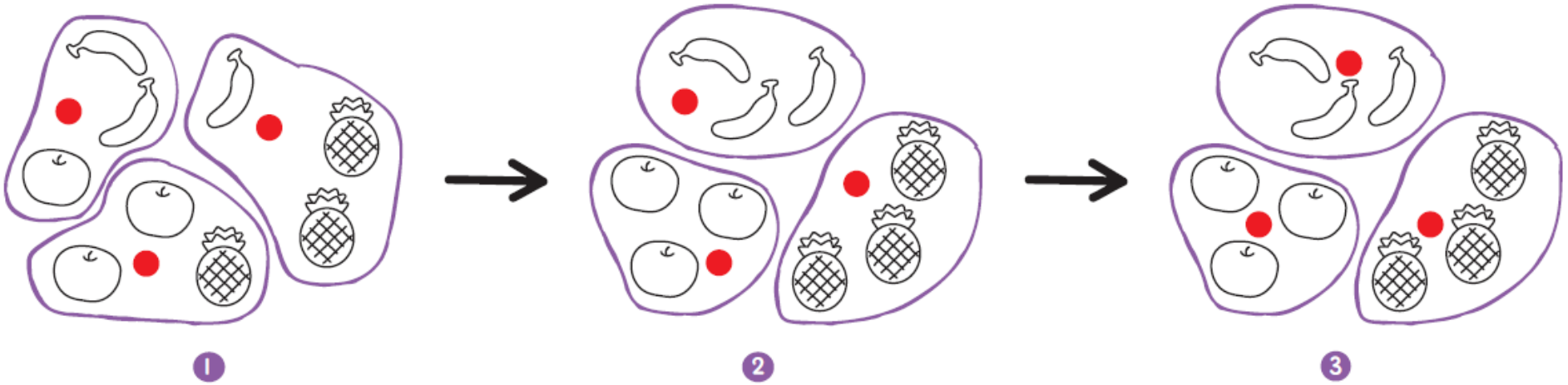


SECTION 6-2 k-평균(2)

◦ k-평균 알고리즘 소개

- k-평균 알고리즘의 작동 방식

1. 무작위로 k개의 클러스터 중심을 정함
2. 각 샘플에서 가장 가까운 클러스터 중심을 찾아 해당 클러스터의 샘플로 지정
3. 클러스터에 속한 샘플의 평균값으로 클러스터 중심을 변경
4. 클러스터 중심에 변화가 없을 때까지 2번으로 돌아가 반복



SECTION 6-2 k-평균(3)

◦ KMeans 클래스

- 1절에서 사용했던 데이터셋을 사용
- wget 명령으로 데이터를 다운로드

```
!wget https://bit.ly/fruits_300 -O fruits_300.npy
```

- np.load( ) 함수를 사용해 npy 파일을 읽어 넘파이 배열을 준비
  - k-평균 모델을 훈련하기 위해 (샘플 개수, 너비, 높이) 크기의 3차원 배열을 (샘플 개수, 너비×높이) 크기를 가진 2차원 배열로 변경

```
import numpy as np
fruits = np.load('fruits_300.npy')
fruits_2d = fruits.reshape(-1, 100*100)
```

- k-평균 알고리즘으로 3개 클러스터로 군집

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3, random_state=42)
km.fit(fruits_2d)
```

SECTION 6-2 k-평균(4)

- KMeans 클래스
  - k-평균의 레이블 출력

```
print(km.labels_)
```

→

```
[0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 2 0 2 0 0 0 0 0 0 2 0 0 0 0 0 0 0 0 2 2 0 0 0 0 0 0 0 0 2 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 2 0 0 0 0 0 0 0 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2
 2 2 2 2 2 2 2 2 2 2 2 2 2 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 2 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1]
```

  - 레이블 0, 1, 2로 모은 샘플의 개수 확인

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

→ (array([0, 1, 2], dtype=int32), array([91, 98, 111]))

SECTION 6-2 k-평균(5)

◦ KMeans 클래스

- 각 클러스터가 어떤 이미지를 나타냈는지 그림으로 출력하기 위해 간단한 유틸리티 함수 draw_fruits( ) 사용

```
import matplotlib.pyplot as plt
def draw_fruits(arr, ratio=1):
    n = len(arr) # n은 샘플 개수입니다
    # 한 줄에 10개씩 이미지를 그립니다. 샘플 개수를 10으로 나누어 전체 행 개수를 계산합니다
    rows = int(np.ceil(n/10))
    # 행이 1개이면 열의 개수는 샘플 개수입니다. 그렇지 않으면 10개입니다
    cols = n if rows < 2 else 10
    fig, axs = plt.subplots(rows, cols,
                            figsize=(cols*ratio, rows*ratio), squeeze=False)
    for i in range(rows):
        for j in range(cols):
            if i*10 + j < n: # n 개까지만 그립니다
                axs[i, j].imshow(arr[i*10 + j], cmap='gray_r')
            axs[i, j].axis('off')
    plt.show()
```

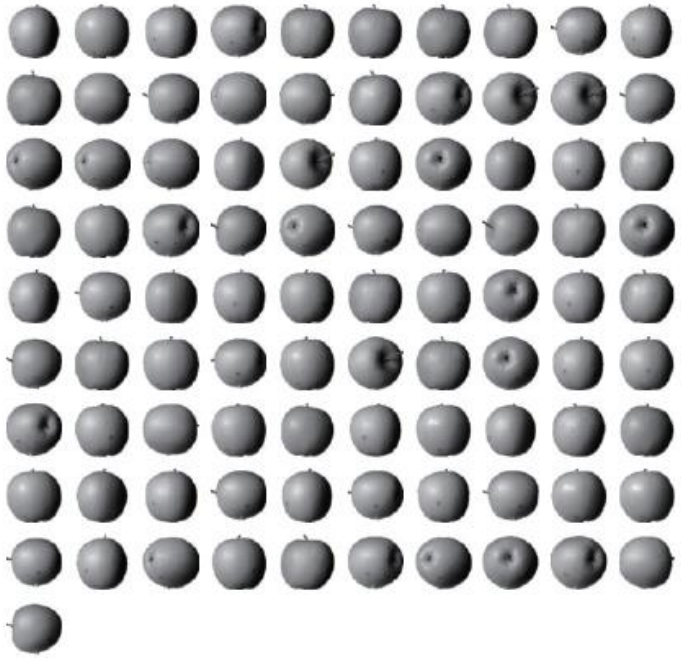
SECTION 6-2 k-평균(6)

◦ KMeans 클래스

- 앞의 함수를 사용해 레이블이 0인 과일 사진을 모두 그리기
  - 불리언 인덱싱 적용하면 True인 위치의 원소만 모두 추출

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

→



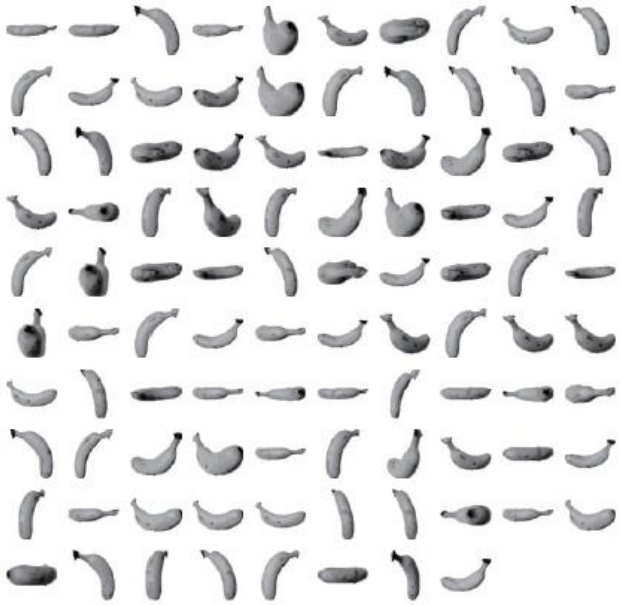
▲ 레이블 0으로 클러스터링된 91개의 이미지를 모두 출력

SECTION 6-2 k-평균(7)

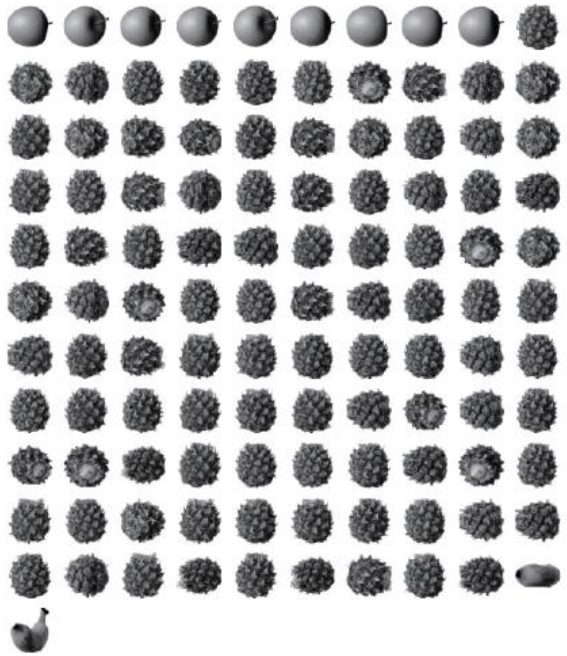
◦ KMeans 클래스

- 앞의 함수를 사용해 레이블이 0인 과일 사진을 모두 그리기
  - 다른 두 클러스터의 이미지 출력

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



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

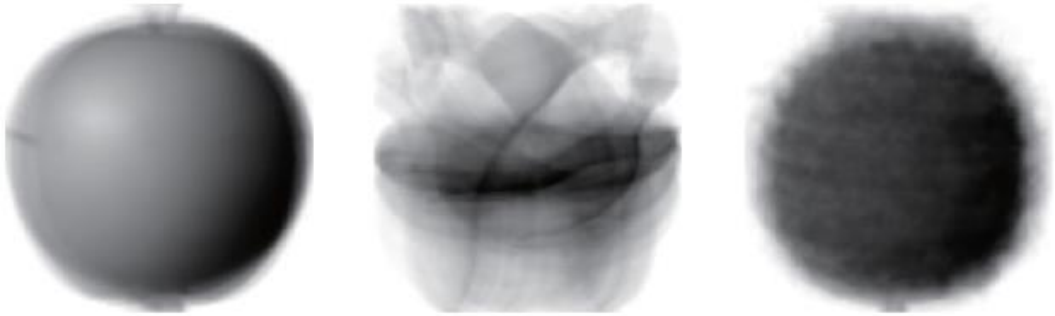


SECTION 6-2 k-평균(8)

◦ 클러스터 중심

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

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



- 훈련 데이터 샘플에서 클러스터 중심까지 거리로 변환해 주는 transform( ) 메서드
- 인덱스가 100인 샘플에 transform( ) 메서드를 적용
  - 슬라이싱 연산자를 사용해서 (1, 10000) 크기의 배열을 전달

```
print(km.transform(fruits_2d[100:101]))
```

→ [[5267.70439881 8837.37750892 3393.8136117]]

SECTION 6-2 k-평균(9)

◦ 클러스터 중심

- 가장 가까운 클러스터 중심을 예측 클래스로 출력하는 predict( ) 메서드

```
print(km.predict(fruits_2d[100:101]))
```
→ [2]

- 클러스터 중심을 그려보았을 때 레이블 2는 파인애플. 이미지로 확인

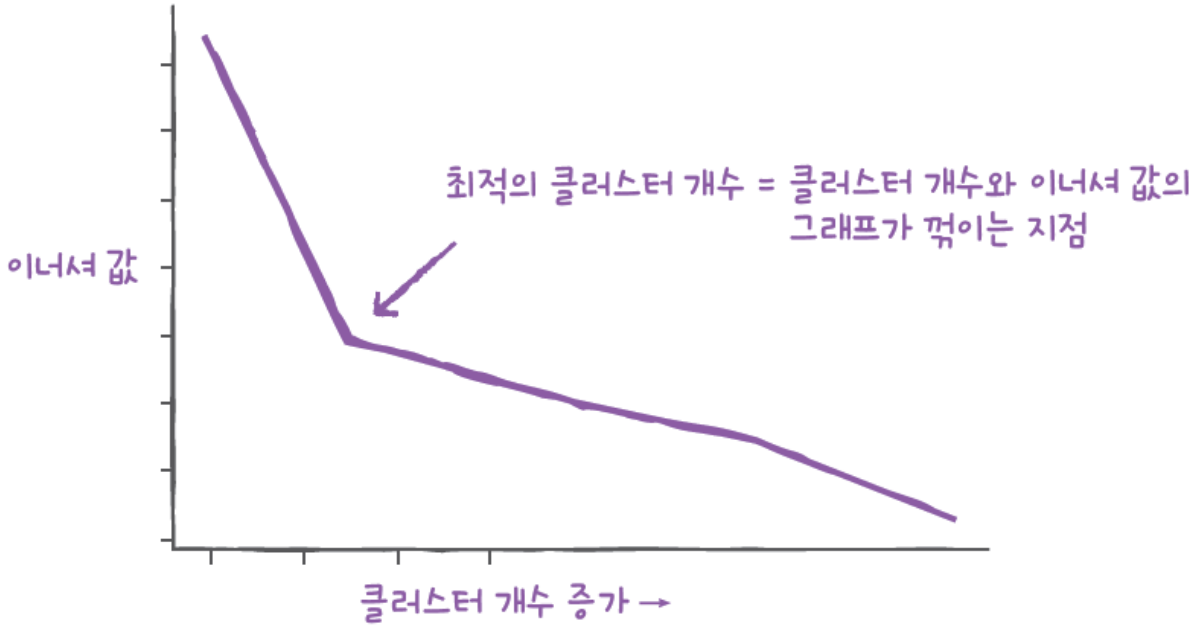
```
draw_fruits(fruits[100:101])
```
→

- 알고리즘이 반복한 횟수는 KMeans 클래스의 n_iter_ 속성에 저장됨

```
print(km.n_iter_)
```
→ 3

SECTION 6-2 k-평균(10)

- 최적의 k 찾기
  - k-평균 알고리즘의 단점 중 하나는 클러스터 개수를 사전에 지정해야 한다는 점
  - 엘보우(elbow) 방법: 적절한 클러스터 개수를 찾기 위한 대표적인 방법
    - 이너셔(inertia): k-평균 알고리즘은 클러스터 중심과 클러스터에 속한 샘플 사이 거리의 제곱 합
    - 이너셔는 클러스터에 속한 샘플이 얼마나 가깝게 모여 있는지를 나타내는 값



SECTION 6-2 k-평균(11)

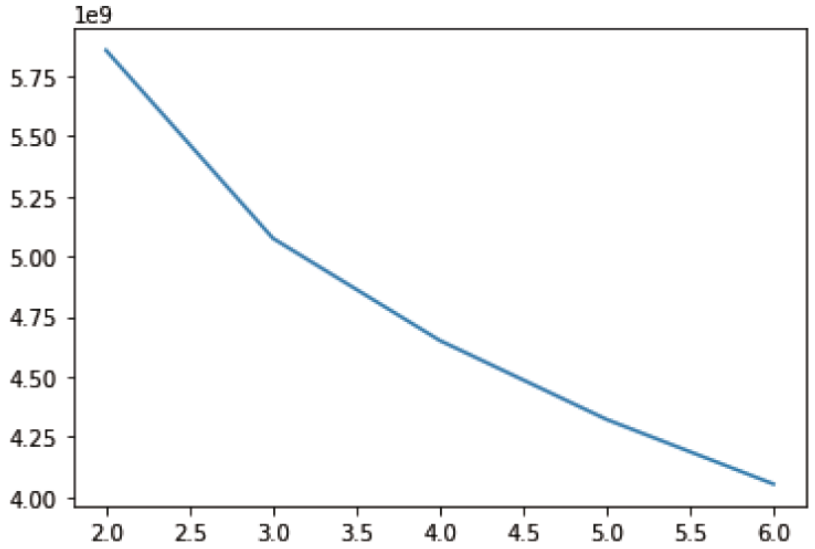
◦ 최적의 k 찾기

▪ 엘보우(elbow) 방법: 적절한 클러스터 개수를 찾기 위한 대표적인 방법

• 과일 데이터셋을 사용해 이너셔 계산하기

```
inertia = []
for k in range(2, 7):
    km = KMeans(n_clusters=k, random_state=42)
    km.fit(fruits_2d)
    Inertia.append(km.inertia_)
plt.plot(range(2, 7), inertia)
plt.show()
```

→



▲ 이 그래프에서는 꺾이는 지점이 뚜렷하지는 않지만, k = 3에서 그래프의 기울기가 조금 바뀐 것을 볼 수 있음
엘보우 지점보다 클러스터 개수가 많아지면 이너셔의 변화가 줄어들면서 군집 효과도 줄어듬
하지만 이 그래프에서는 이런 지점이 명확하지는 않음

SECTION 6-2 k-평균(12)

◦ 과일을 자동으로 분류하기(문제해결 과정)

- 문제
  - 타깃값을 모르는 척하고 자동으로 사진을 클러스터로 모을 수 있는 군집 알고리즘이 필요
- 해결
  - k-평균 알고리즘
  - transform( ) 메서드: 각 샘플에서 각 클러스터까지의 거리를 하나의 특성으로 활용
  - predict( ) 메서드에서 새로운 샘플에 대해 가장 가까운 클러스터를 예측값으로 출력
  - 최적의 클러스터 개수 k를 알아내기 위해 클러스터가 얼마나 밀집되어 있는지 나타내는 이너셔를 사용
  - 이너셔가 더 이상 크게 줄어들지 않는다면 클러스터 개수를 더 늘리는 것은 효과가 없음
  - 엘보우 방법: 클러스터 개수를 늘리면서 반복하여 KMeans 알고리즘을 훈련하고 이너셔가 줄어드는 속도가 꺾이는 지점을 최적의 클러스터 개수로 결정

SECTION 6-2 마무리(1)

- 키워드로 끝나는 핵심 포인트
  - k-평균 알고리즘은 처음에 랜덤하게 클러스터 중심을 정하고 클러스터를 만듦
    - 클러스터의 중심을 이동하고 다시 클러스터를 만드는 식으로 반복해서 최적의 클러스터를 구성하는 알고리즘
  - 클러스터 중심은 k-평균 알고리즘이 만든 클러스터에 속한 샘플의 특성 평균값
    - 센트로이드(centroid)라고도 부름
    - 가장 가까운 클러스터 중심을 샘플의 또 다른 특성으로 사용하거나 새로운 샘플에 대한 예측으로 활용
  - 엘보우 방법은 최적의 클러스터 개수를 정하는 방법 중 하나
    - 이너셔는 클러스터 중심과 샘플 사이 거리의 제곱 합
    - 클러스터 개수에 따라 이너셔 감소가 꺾이는 지점이 적절한 클러스터 개수 k가 될 수 있음
    - 이 그래프의 모양을 따서 엘보우 방법이라고 부름

SECTION 6-2 마무리(2)

- 핵심 패키지와 함수
  - scikit-learn
    - Kmeans: k-평균 알고리즘 클래스

SECTION 6-2 확인 문제

1. k-평균 알고리즘에서 클러스터를 표현하는 방법이 아닌것은?

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

2. k-평균에서 최적의 클러스터 개수는 어떻게 정할 수 있나?

   ① 엘보우 방법을 사용해 이너셔의 감소 정도가 꺾이는 클러스터 개수를 찾는다
   ② 랜덤하게 클러스터 개수를 정해서 k-평균 알고리즘을 훈련하고 가장 낮은 이너셔가 나오는 클러스터 개수를 찾는다
   ③ 훈련 데이터를 모두 조사하여 몇 개의 클러스터가 나올 수 있는지 직접 확인
   ④ 교차 검증을 사용하여 최적의 클러스터 개수를 찾는다

SECTION 6-3 주성분 분석(1)

◦ 차원과 차원 축소

- 특성은 데이터가 가진 속성. 과일 사진의 경우 10,000개의 픽셀은 10,000개의 특성이 있는 셈
- 머신러닝에서는 이런 특성을 차원(dimension)이라고도 함
- 10,000개의 특성은 결국 10,000개의 차원이라는 건데 이 차원을 줄일 수 있다면 저장 공간을 크게 절약

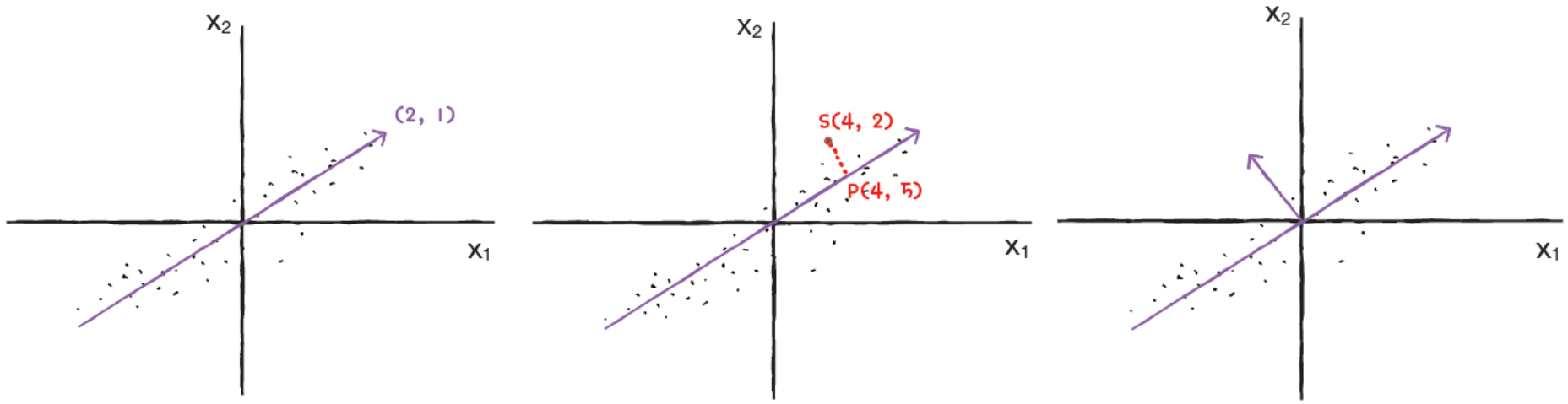
▪ 차원 축소(dimensionality reduction) 알고리즘

- 차원 축소는 데이터를 가장 잘 나타내는 일부 특성을 선택하여 데이터 크기를 줄이고 지도 학습 모델의 성능을 향상시킬 수 있는 방법
- 또한 줄어든 차원에서 다시 원본 차원(예를 들어 과일 사진의 경우 10,000개의 차원)으로 손실을 최대한 줄이면서 복원할 수도 있음
- 주성분 분석(principal component analysis, PCA)은 대표적인 차원 축소 알고리즘

SECTION 6-3 주성분 분석(2)

◦ 주성분 분석 소개

- 주성분 분석(PCA)은 데이터에 있는 분산이 큰 방향을 찾는 것
- 분산이 큰 방향을 찾은 직선이 원점에서 출발한다면 두 원소로 이루어진 벡터로 표현 가능
- 주성분 벡터의 원소 개수는 원본 데이터셋에 있는 특성 개수와 같음
- 원본 데이터는 주성분을 사용해 차원을 줄일 수 있음
- 주성분은 원본 차원과 같고 주성분으로 바꾼 데이터는 차원이 줄어듦



▲ [노트]실제로 사이킷런의 PCA 모델을 훈련하면 자동으로 특성마다 평균값을 빼서 원점에 맞춰 줌. 따라서 우리가 수동으로 데이터를 원점에 맞출 필요가 없음

SECTION 6-3 주성분 분석(3)

- PCA 클래스
  - 이전 절과 마찬가지로 과일 사진 데이터를 다운로드하여 넘파이 배열로 적재

```
!wget https://bit.ly/fruits_300 -O fruits_300.npy
import numpy as np
fruits = np.load('fruits_300.npy')
fruits_2d = fruits.reshape(-1, 100*100)
```

  - PCA 클래스로 주성분 분석

```
from sklearn.decomposition import PCA
pca = PCA(n_components=50)
pca.fit(fruits_2d)
```

  - 배열의 크기 확인

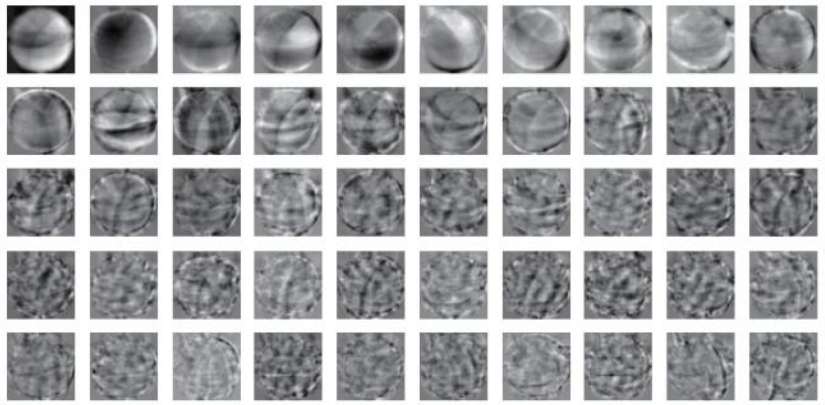
```
print(pca.components_.shape)
```

→ (50, 10000)

  - draw_fruits( ) 함수를 사용해서 주성분 그리기

```
draw_fruits(pca.components_.reshape(-1, 100, 100))
```

→



SECTION 6-3 주성분 분석(4)

◦ PCA 클래스

- 원본 데이터를 주성분에 투영하여 특성의 개수를 10,000개에서 50개로 줄일 수 있음
- PCA의 transform( ) 메서드를 사용해 원본 데이터의 차원을 50으로 축소

```
print(fruits_2d.shape)
```

→ (300, 10000)

```
fruits_pca = pca.transform(fruits_2d)
print(fruits_pca.shape)
```

→ (300, 50)

▲ 데이터가 1/200으로 축소

SECTION 6-3 주성분 분석(5)

◦ 원본 데이터 재구성

- inverse_transform( ) 메서드를 사요애, 앞서 50개의 차원으로 축소한 fruits_pca 데이터를 전달해 10,000개의 특성을 복원

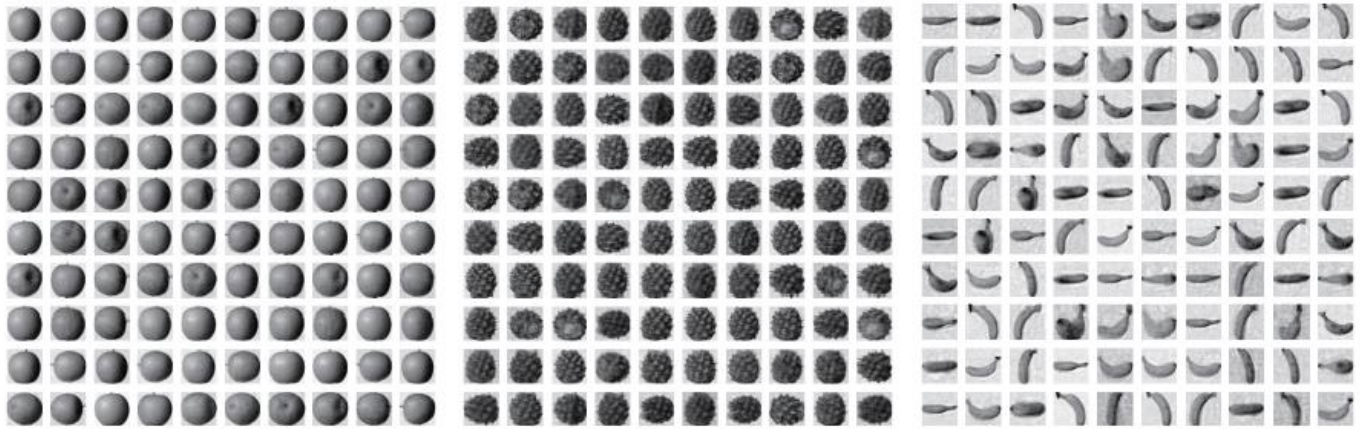
```
fruits_inverse = pca.inverse_transform(fruits_pca)
print(fruits_inverse.shape)
```

→ (300, 10000)

- 100 × 100 크기로 바꾸어 100개씩 나누어 출력

```
fruits_reconstruct = fruits_inverse.reshape(-1, 100, 100)
for start in [0, 100, 200]:
    draw_fruits(fruits_reconstruct[start:start+100])
    print("\n")
```

→



SECTION 6-3 주성분 분석(6)

◦ 설명된 분산(explained variance)

- 주성분이 원본 데이터의 분산을 얼마나 잘 나타내는지 기록한 값
- PCA 클래스의 explained_variance_ratio_에 각 주성분의 설명된 분산 비율이 기록
  - 첫 번째 주성분의 설명된 분산이 가장 큼
  - 분산 비율을 모두 더하면 50개의 주성분으로 표현하고 있는 총 분산 비율을 얻을 수 있음

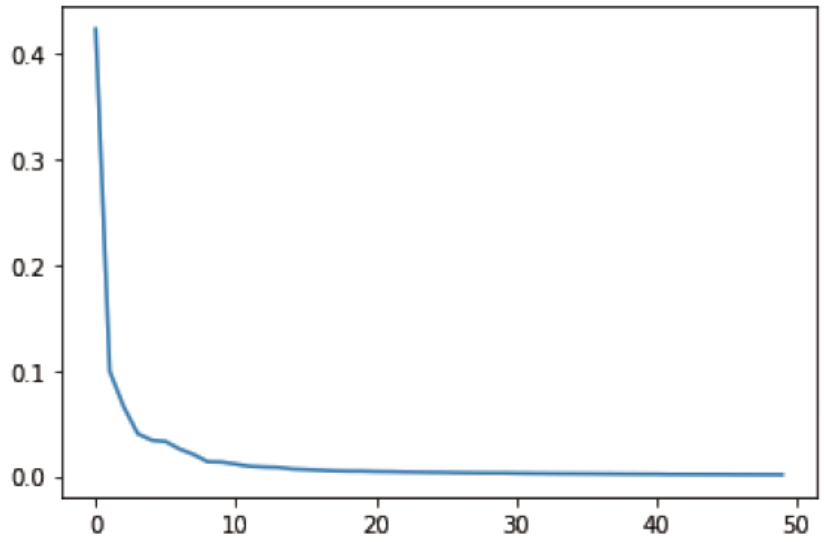
```
print(np.sum(pca.explained_variance_ratio_))
```

→ 0.9215190262621741

- 맷플롯립의 plot( ) 함수로 설명된 분산을 그래프로 출력

```
plt.plot(pca.explained_variance_ratio_)
plt.show()
```

→



SECTION 6-3 주성분 분석(7)

◦ 다른 알고리즘과 함께 사용하기

- 과일 사진 원본 데이터와 PCA로 축소한 데이터를 지도 학습에 적용해 보고 차이점 관찰
- 3개의 과일 사진을 분류를 위해 사이킷런의 LogisticRegression 모델 만들기

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
```

- 사과를 0, 파인애플을 1, 바나나를 2로 지정하여, 100개의 0, 100개의 1, 100개의 2로 이루어진 타깃 데이터 만들기

```
target = np.array([0]*100 + [1]*100 + [2]*100)
```

- cross_validate( )로 교차 검증 수행

```
from sklearn.model_selection import cross_validate
scores = cross_validate(lr, fruits_2d, target)
print(np.mean(scores['test_score']))
print(np.mean(scores['fit_time']))
```

→
0.9966666666666667
0.9422160625457764

- PCA로 축소한 fruits_pca를 사용했을 때와 비교

```
scores = cross_validate(lr, fruits_pca, target)
print(np.mean(scores['test_score']))
print(np.mean(scores['fit_time']))
```

→
1.0
0.03256878852844238

▲ 50개의 특성만 사용했는데도 정확도가 100%이고 훈련 시간은 0.03초로 20배 이상 감소

SECTION 6-3 주성분 분석(8)

◦ 다른 알고리즘과 함께 사용하기

- n_components 매개변수에 분산의 비율을 입력하여. 설명된 분산의 50%에 달하는 주성분을 찾도록 PCA 모델 만들기

```
pca = PCA(n_components=0.5)
pca.fit(fruits_2d)
```

- 탐색한 주성분 개수 확인

```
print(pca.n_components_)
```

→ 2

- 이 모델로 원본 데이터를 변환

```
fruits_pca = pca.transform(fruits_2d)
print(fruits_pca.shape)
```

→ (300, 2)

- 교차 검증의 결과 확인

```
scores = cross_validate(lr, fruits_pca, target)
print(np.mean(scores['test_score']))
print(np.mean(scores['fit_time']))
```

→ 0.9933333333333334
0.04122166633605957

SECTION 6-3 주성분 분석(9)

- 다른 알고리즘과 함께 사용하기
  - 차원 축소된 데이터를 사용해 k-평균 알고리즘으로 클러스터 찾기

```
from sklearn.cluster import KMeans
km = KMeans(n_clusters=3, random_state=42)
km.fit(fruits_pca)
print(np.unique(km.labels_, return_counts=True))
```

→ (array([0, 1, 2], dtype=int32), array([91, 99, 110]))

  - KMeans가 찾은 레이블을 사용해 과일 이미지 출력

```
for label in range(0, 3):
    draw_fruits(fruits[km.labels_ == label])
    print("\n")
```

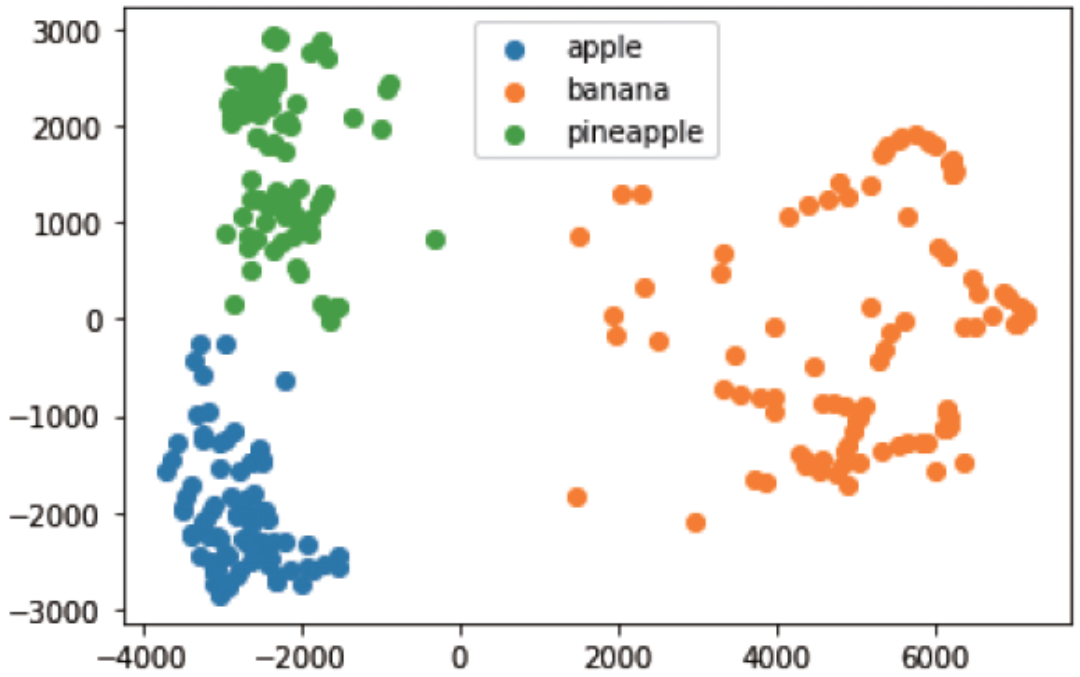
→

SECTION 6-3 주성분 분석(10)

- 다른 알고리즘과 함께 사용하기
  - 훈련 데이터의 차원을 줄이면 또 하나 얻을 수 있는 장점은 시각화
  - 3개 이하로 차원을 줄이면 화면에 출력하기 비교적 쉬움
  - fruits_pca 데이터는 2개의 특성이 있기 때문에 2차원으로 표현
  - 앞에서 찾은 km.labels_를 사용해 클러스터별로 나누어 산점도 그리기

```
for label in range(0, 3):
    data = fruits_pca[km.labels_ == label]
    plt.scatter(data[:,0], data[:,1])
plt.legend(['apple', 'banana', 'pineapple'])
plt.show()
```

→



SECTION 6-3 주성분 분석(11)

- 주성분 분석으로 차원 축소(문제해결 과정)
  - 대표적인 비지도 학습 문제 중 하나인 차원 축소
    - 차원 축소를 사용하면 데이터셋의 크기를 줄일 수 있고 비교적 시각화하기 쉬움
    - 또 차원 축소된 데이터를 지도 학습 알고리즘이나 다른 비지도 학습 알고리즘에 재사용하여 성능을 높이거나 훈련 속도를 빠르게 만들 수 있음
  - 학습
    - 사이킷런의 PCA 클래스를 사용해 과일 사진 데이터의 특성을 50개로 크게 축소
    - 특성 개수는 작지만 변환된 데이터는 원본 데이터에 있는 분산의 90% 이상을 표현 (이를 설명된 분산이라 함)
    - PCA 클래스는 자동으로 설명된 분산을 계산하여 제공하고, 주성분의 개수를 명시적으로 지정하는 대신 설명된 분산의 비율을 설정하여 원하는 비율만큼 주성분을 찾을 수 있음
    - PCA 클래스는 변환된 데이터에서 원본 데이터를 복원하는 메서드도 제공
    - 변환된 데이터가 원본 데이터의 분산을 모두 유지하고 있지 않다면 완벽하게 복원되지 않지만 적은 특성으로도 상당 부분의 디테일을 복원할 수 있음

SECTION 6-3 마무리(1)

- 키워드로 끝나는 핵심 포인트
  - 차원 축소는 원본 데이터의 특성을 적은 수의 새로운 특성으로 변환하는 비지도 학습의 한 종류
    - 차원 축소는 저장 공간을 줄이고 시각화하기 쉬움
    - 또한 다른 알고리즘의 성능을 높일 수도 있음
  - 주성분 분석은 차원 축소 알고리즘의 하나로 데이터에서 가장 분산이 큰 방향을 찾는 방법
    - 이런 방향을 주성분이라고 함
    - 원본 데이터를 주성분에 투영하여 새로운 특성을 만들 수 있음
    - 일반적으로 주성분은 원본 데이터에 있는 특성 개수보다 적음
  - 설명된 분산은 주성분 분석에서 주성분이 얼마나 원본 데이터의 분산을 잘 나타내는지 기록한 것
    - 사이킷런의 PCA 클래스는 주성분 개수나 설명된 분산의 비율을 지정하여 주성분 분석을 수행할 수 있음

SECTION 6-3 마무리(2)

- 핵심 패키지와 함수
  - scikit-learn
    - PCA: 주성분 분석을 수행하는 클래스
      - n_components: 주성분의 개수를 지정. 기본값은 None으로 샘플 개수와 특성 개수 중에 작은 것의 값을 사용
      - random_state: 넘파이 난수 시드 값을 지정
      - components_: 훈련 세트에서 찾은 주성분이 저장
      - explained_variance_: 설명된 분산이 저장
      - explained_variance_ratio_: 설명된 분산의 비율이 저장
      - inverse_transform( ) 메서드: transform( ) 메서드로 차원을 축소시킨 데이터를 다시 원본 차원으로 복원

SECTION 6-3 확인 문제

1. 샘플 개수가 1,000개이고 특성 개수는 100개인 데이터셋이 있다. 즉 이 데이터셋의 크기는 (1000, 100). 이 데이터를 사이킷런의 PCA 클래스를 사용해 10개의 주성분을 찾아 변환했을 때, 변환된 데이터셋의 크기는 얼마일까?

   ① (1000, 10)
   ② (10, 1000)
   ③ (10, 10)
   ④ (1000, 1000)

2. 위 문제에서 설명된 분산이 가장 큰 주성분은 몇 번째인가?

   ① 첫 번째 주성분
   ② 다섯 번째 주성분
   ③ 열 번째 주성분
   ④ 알 수 없음