



인공신경망과딥러닝입문

Lecture 15. Principal Component Analysis

주

성분

분석

동덕여자대학교
데이터사이언스 전공
권 범

목차

❖ 01. 차원과 차원 축소

❖ 02. PCA 소개 $\xrightarrow{\text{목적}}$ ① 차원 축소 (Dimensionality Reduction) ↳ 데이터 압축 → 저장 공간 확보

❖ 03. PCA 클래스

② 잡음 제거 (Noise Elimination)

❖ 04. 다른 알고리즘과 함께 사용하기



❖ 05. PCA로 차원 축소

❖ 06. 마무리

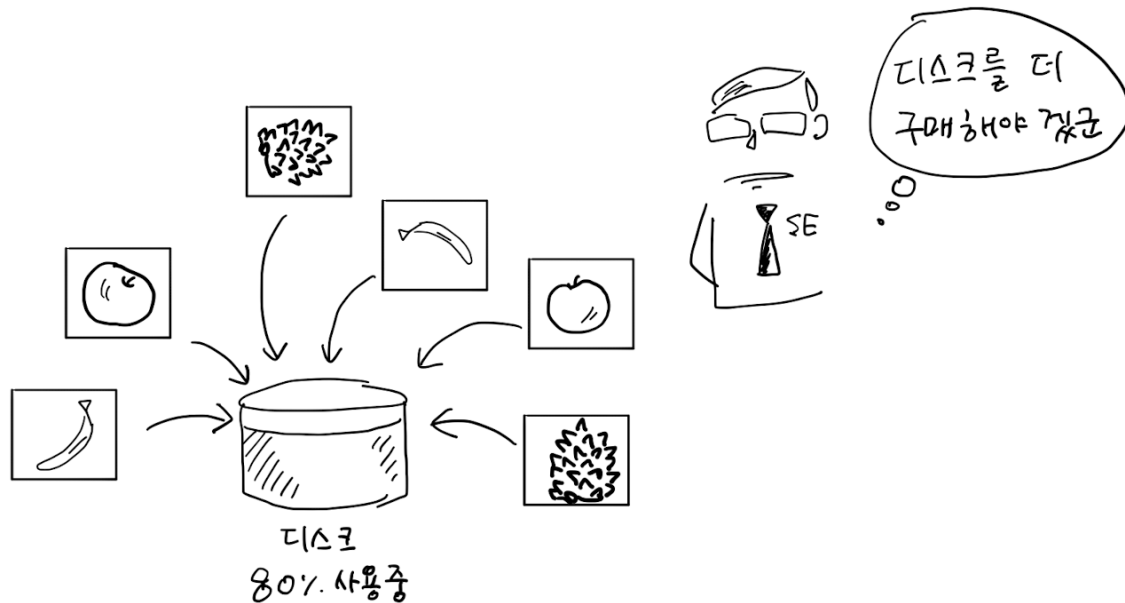
01. 차원과 차원 축소

- 02. PCA 소개
- 03. PCA 클래스
- 04. 다른 알고리즘과 함께 사용하기
- 05. PCA로 차원 축소
- 06. 마무리

01. 차원과 차원 축소

❖ 시작하기 전에

- 동덕 마켓의 과일 사진 이벤트가 대성공임
- 매일 각양각색의 과일 사진이 업로드되고 있음
- k-Means Clustering Algorithm으로 업로드된 사진을 클러스터로 분류하려 폴더별로 저장하였음
- 그런데 이벤트가 진행되면서 문제가 생겼음
- 너무 많은 사진이 등록되어 **저장 공간이 부족함**



(1470장)
fruits.npy → 56 MB

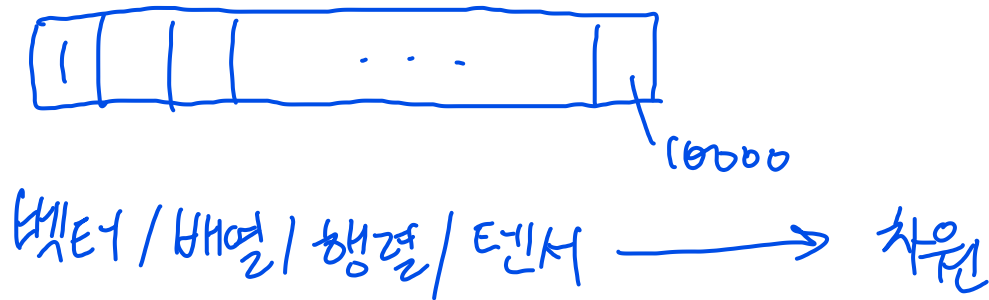
업로드된 사진의 용량을
줄일 수 있는 방법은 없을까?

01. 차원과 차원 축소

❖ 차원과 차원 축소 (1/3)

- 지금까지 우리는 데이터가 가진 속성을 특성(Feature)이라고 불렀음
- 과일 사진의 경우 10,000개의 픽셀이 있기 때문에 10,000개의 특성이 있는 셈임
- 머신러닝에서는 이런 특성을 차원(Dimension)이라고 부름

10,000개의 특성은 결국 10,000개의 차원이라는 건데
이 차원을 줄일 수 있다면 저장 공간을 크게 절약할 수 있을 것임

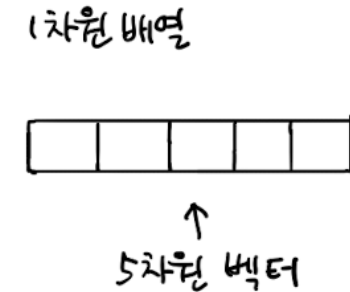
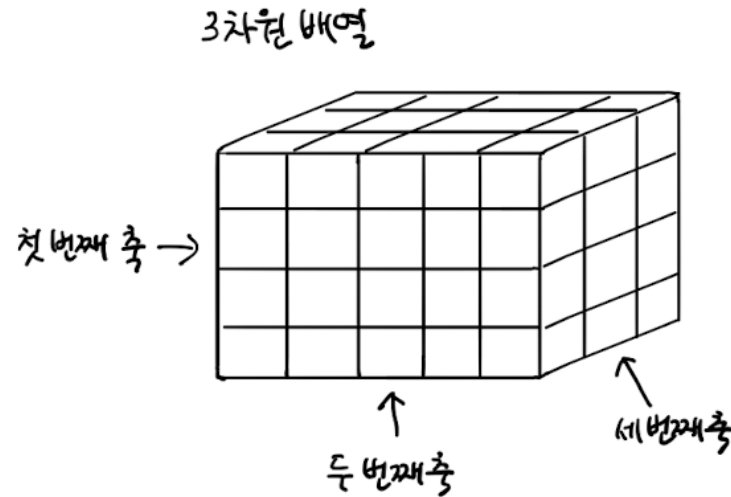
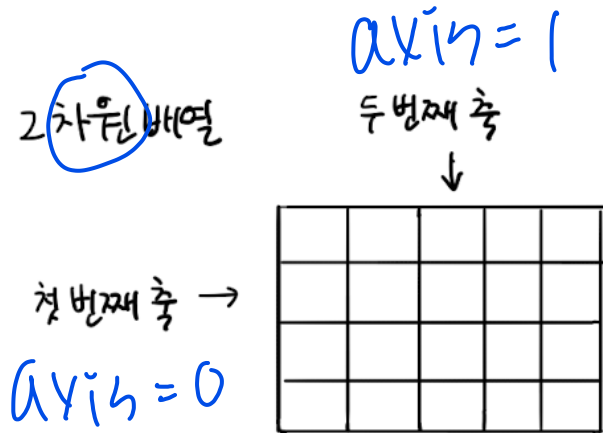


01. 차원과 차원 축소

❖ 차원과 차원 축소 (2/3)

2차원 배열과 1차원 배열의 차원은 다른 것인가요?

- ✓ 2차원 배열과 1차원 배열(벡터, Vector)에서 차원이라는 용어는 조금 다르게 사용함
- ✓ 다차원 배열에서 차원은 배열의 축 개수가 됨
- ✓ 가령 2차원 배열일 때는 행(Row)과 열(Column)이 차원이 됨
- ✓ 하지만 1차원 배열, 즉 벡터일 경우에는 원소의 개수를 말함



01. 차원과 차원 축소

❖ 차원과 차원 축소 (3/3)

- 이번 수업 시간에는 비지도 학습 작업 중 하나인 차원 축소 알고리즘을 다루어 보겠음
- 차원 축소는 데이터를 가장 잘 나타내는 일부 특성을 선택하여 데이터 크기를 줄이는 방법임
- 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)

대표적인 차원 축소 알고리즘임.
주성분 분석을 간단히 PCA라고도 부름

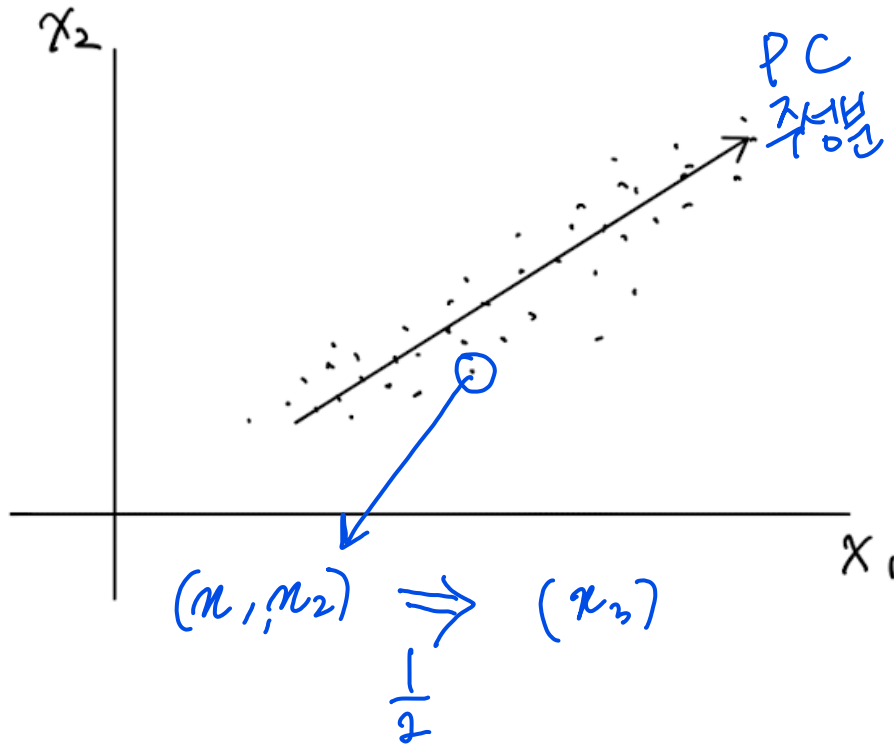
02. PCA 소개

- 01. 차원과 차원 축소
- 03. PCA 클래스
- 04. 다른 알고리즘과 함께 사용하기
- 05. PCA로 차원 축소
- 06. 마무리

02. PCA 소개

❖ PCA 개념 이해 (1/4)

- PCA는 데이터에 있는 **분산**(Variance)이 큰 방향을 찾는 것으로 이해할 수 있음
- 분산은 데이터가 널리 퍼져있는 정도를 말함
- 분산이 큰 방향이란 데이터를 잘 표현하는 어떤 벡터라고 생각할 수 있음

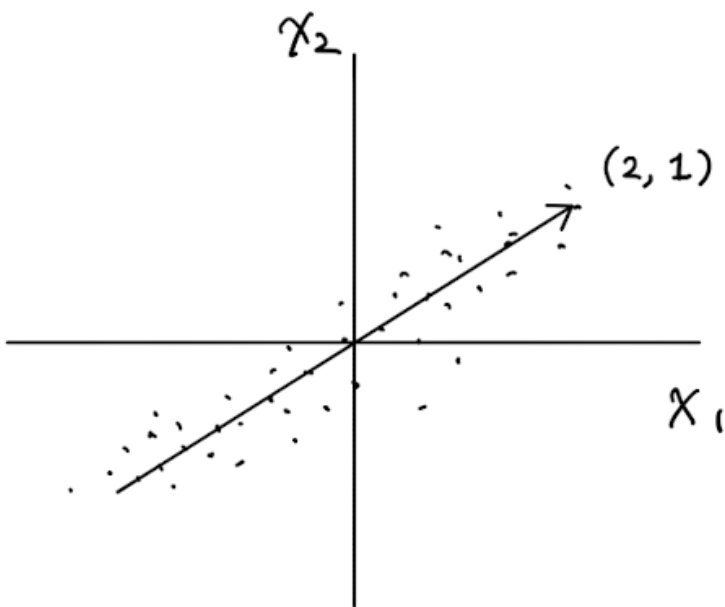


- ✓ 이 데이터는 x_1, x_2 2개의 특성이 있음
- ✓ 대각선 방향으로 길게 늘어진 형태를 가지고 있음
- ✓ 직관적으로 우리는 길게 늘어진 대각선 방향이 분산이 가장 크다고 알 수 있음
- ✓ 화살표 위치는 큰 의미가 없음
- ✓ 오른쪽 위로 향하거나 왼쪽 아래로 향할 수도 있음
- ✓ 중요한 것은 분산이 큰 방향을 찾는 것임

02. PCA 소개

❖ PCA 개념 이해 (2/4)

- 앞에서 찾은 직선이 원점에서 출발한다면 두 원소로 이루어진 벡터로 쓸 수 있음
- 이 벡터를 Principal Component(주성분)라고 부름
- 이 주성분 벡터는 원본 데이터에 있는 어떤 방향임
- 따라서 주성분 벡터의 원소 개수는 원본 데이터셋에 있는 특성 개수와 같음



- ✓ 실제로 scikit-learn의 PCA 모델을 학습시키면 자동으로 특성마다 평균값을 빼서 원점에 맞춰 줌
- ✓ 따라서 우리가 수동으로 데이터를 원점에 맞추는 필요가 없음

02. PCA 소개

❖ PCA 개념 이해 (3/4)

- 원본 데이터는 주성분을 사용해 차원을 줄일 수 있음
- 예를 들면 샘플 데이터 $s(4, 2)$ 를 주성분에 직각으로 투영하면 1차원 데이터 $p(4.5)$ 를 만들 수 있음



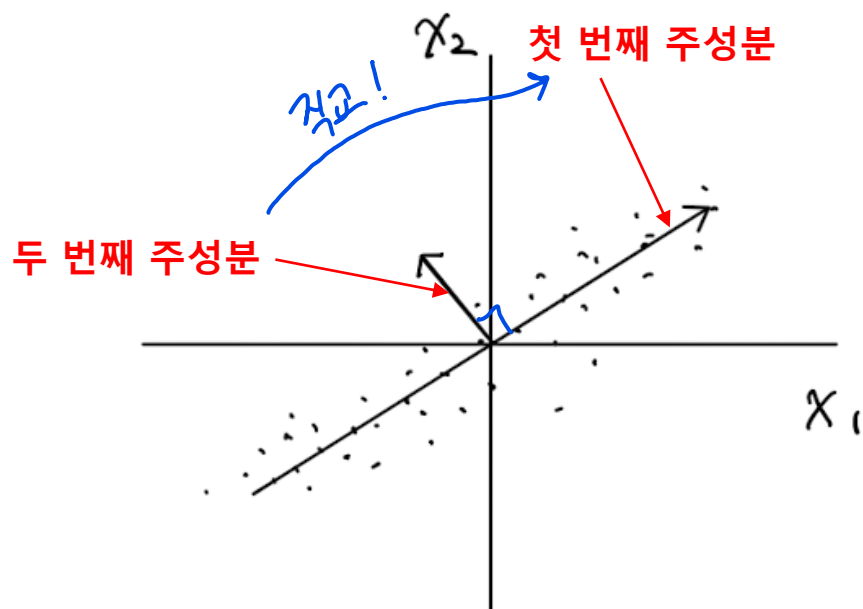
주성분은 원본 차원과 같고 주성분으로 바꾼 데이터는 차원이 줄어든다는 점을 꼭 기억하자

주성분이 가장 분산이 큰 방향이기 때문에
주성분에 투영하여 바꾼 데이터는
원본이 가지고 있는 특성을 가장 잘 나타내고 있을 것임

02. PCA 소개

❖ PCA 개념 이해 (4/4)

- 첫 번째 주성분을 찾은 다음, 이 벡터에 수직이고 분산이 가장 큰 다음 방향을 찾음
- 찾은 벡터가 두 번째 주성분이 됨
- 이 예제에서는 2차원이기 때문에 두 번째 주성분은 다음처럼 하나뿐임
- 일반적으로 주성분은 원본 특성의 개수만큼 찾을 수 있음



- ✓ 주성분은 원본 특성의 개수와 샘플 개수 중 작은 값만큼 찾을 수 있음
- ✓ 일반적으로 비지도 학습은 대량의 데이터에서 수행하기 때문에 원본 특성의 개수만큼 찾을 수 있다고 말함

이제 scikit-learn으로 과일 사진 데이터에서
PCA를 실습해 보겠음



03. PCA 클래스

- 01. 차원과 차원 축소
- 02. PCA 소개
- 04. 다른 알고리즘과 함께 사용하기
- 05. PCA로 차원 축소
- 06. 마무리

03. PCA 클래스

❖ ① 실습 Dataset 준비하기 (1/2)

- 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

03. PCA 클래스

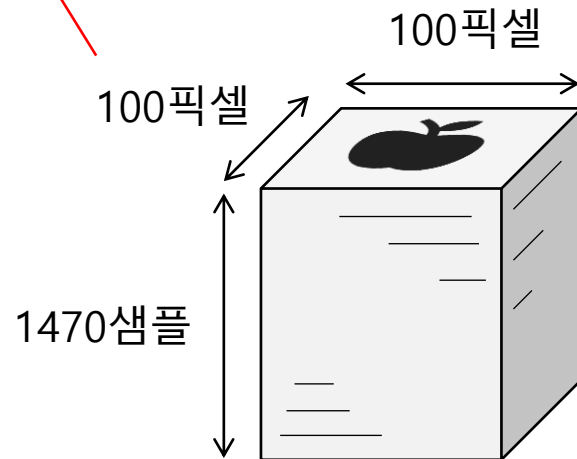
❖ ① 실습 Dataset 준비하기 (2/3)

- fruit_npy 배열의 shape 속성을 확인해 보자

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

실행결과

(1470, 100, 100)

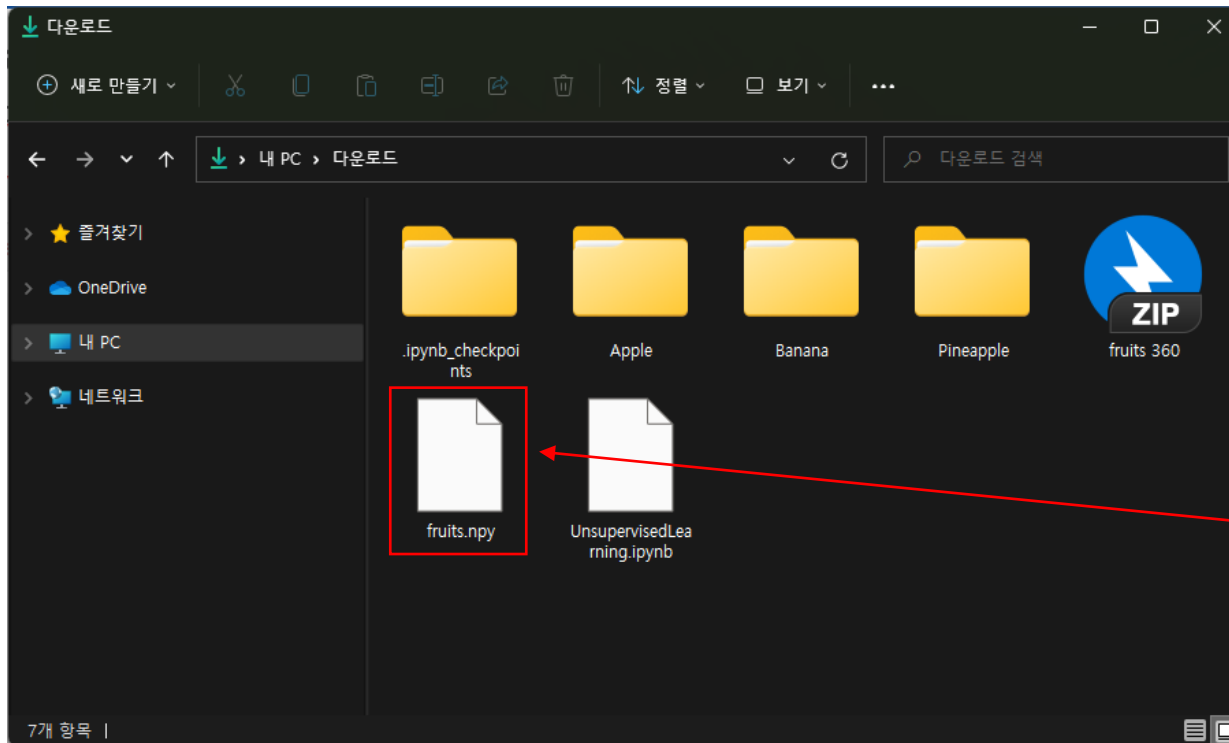


03. PCA 클래스

❖ 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

03. PCA 클래스

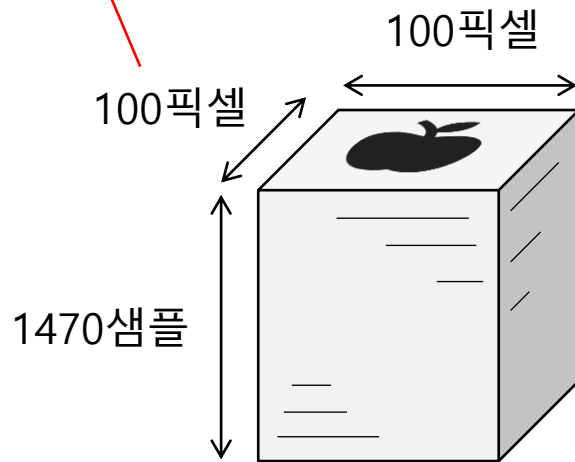
❖ ② npy 파일 로드하기

- 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

PCA 클래스의 fit() 메서드에 값을 전달하기 위해서는 배열의 크기를 변경해야 함

03. PCA 클래스

❖ ③ 3차원 배열을 2차원 배열로 변경하기

- PCA 모델을 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)

03. PCA 클래스

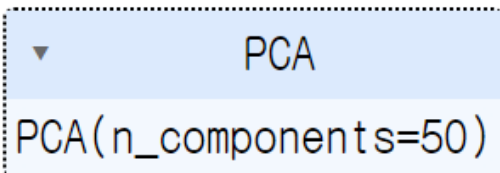
❖ ④ PCA 클래스의 객체를 생성하고 학습시키기 (1/2)

- scikit-learn의 sklearn.decomposition 모듈 아래 PCA 클래스에 구현되어 있음
- PCA 클래스의 객체를 만들 때 n_components 매개변수에 주성분의 개수를 지정해야 함
- k-Means Clustering Algorithm과 마찬가지로 비지도 학습이기 때문에 fit() 메서드에서 타겟값을 전달하지 않음

```
1 from sklearn.decomposition import PCA
2
3 pca = PCA(n_components=50)   주성분의 개수 n_components를 50으로 지정하겠음
4
5 pca.fit(fruits_2d)
```

- decompose
(동사) 분해되다
(동사) 분해하다
- decomposition
(명사) 분해

실행결과



```
PCA(n_components=50)
```

03. PCA 클래스

❖ ④ PCA 클래스의 객체를 생성하고 학습시키기 (2/2)

- PCA 클래스가 찾은 주성분은 `components_` 속성에 저장되어 있음
- 이 배열의 크기를 확인하겠음

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

실행결과

```
(50, 10000)
```

- ✓ `n_components=50`으로 지정했기 때문에 `pca.components_` 배열의 첫 번째 차원이 50임
- ✓ 즉 50개의 주성분을 찾은 것임
- ✓ 두 번째 차원은 항상 원본 데이터의 특성 개수와 같은 10,000임

- `component`
(명사) 요소
(명사) 부품

03. PCA 클래스

❖ ⑤ 주성분을 이미지로 확인하기 (1/2)

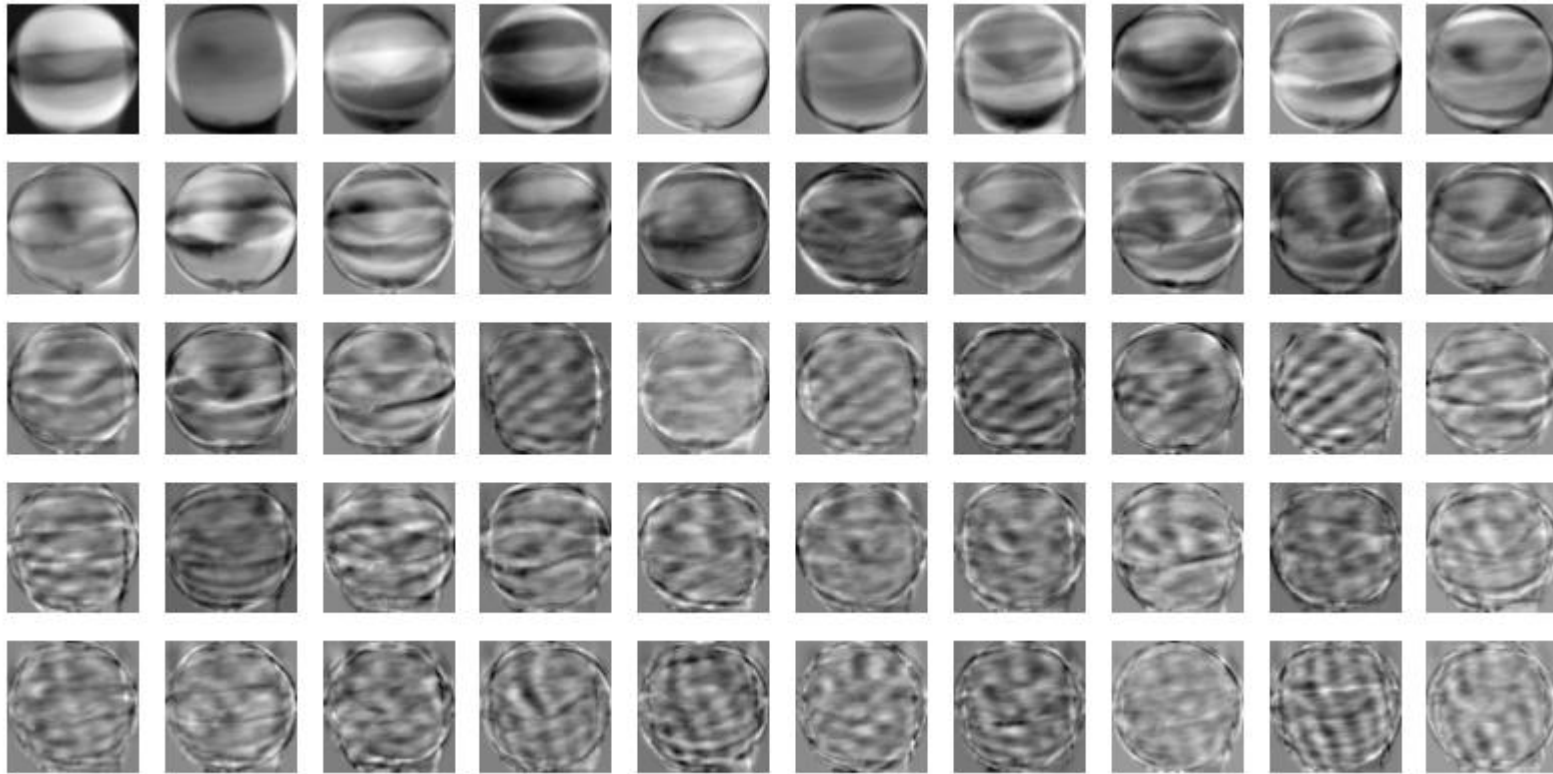
- 원본 데이터와 차원이 같으므로 주성분을 100×100 크기의 이미지처럼 출력해 볼 수 있음
- k-Means Clustering Algorithm 수업 때 사용했던 draw_fruits() 함수를 사용해서 이 주성분을 그림으로 그려보자

```
1 def draw_fruits(arr, ratio=1):
2     n = len(arr)
3     rows = int(np.ceil(n/10))
4     cols = n if rows < 2 else 10
5
6     fig, axs = plt.subplots(rows, cols, figsize=(cols * ratio, rows * ratio),
7                               squeeze=False)
8     for j in range(rows):
9         for k in range(cols):
10             if j * 10 + k < n:
11                 axs[j, k].imshow(arr[j * 10 + k], cmap="gray_r")
12                 axs[j, k].axis("off")
13     plt.show()
14
15
16 draw_fruits(pca.components_.reshape(-1, 100, 100))
```

03. PCA 클래스

❖ ⑤ 주성분을 이미지로 확인하기 (2/2)

실행결과



- ✓ 이 주성분은 원본 데이터에서 가장 분산이 큰 방향을 순서대로 나타낸 것임
- ✓ 한편으로는 데이터셋에 있는 어떤 특징을 잡아낸 것처럼 생각할 수도 있음

03. PCA 클래스

❖ ⑥ 원본 데이터의 차원 줄이기

- 주성분을 찾았으므로 원본 데이터를 주성분에 투영하여 특성의 개수를 10,000개에서 50개로 줄일 수 있음
- 이는 마치 원본 데이터를 각 주성분으로 분해하는 것으로 생각할 수 있음
- PCA의 transform() 메서드를 사용해 원본 데이터의 차원을 50으로 줄여 보겠음

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

실행결과

```
(1470, 10000)
```

- ✓ fruits_2d는 (1470, 10000) 크기의 배열임
- ✓ 10,000개의 픽셀(특성)을 가진 1,470개의 이미지임

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

실행결과

```
(1470, 50)
```

- ✓ 50개의 주성분을 찾은 PCA 모델을 사용해 이를 (1470, 50) 크기의 배열로 변환함
- ✓ 이제 fruits_pca 배열은 50개의 특성을 가진 데이터임

03. PCA 클래스

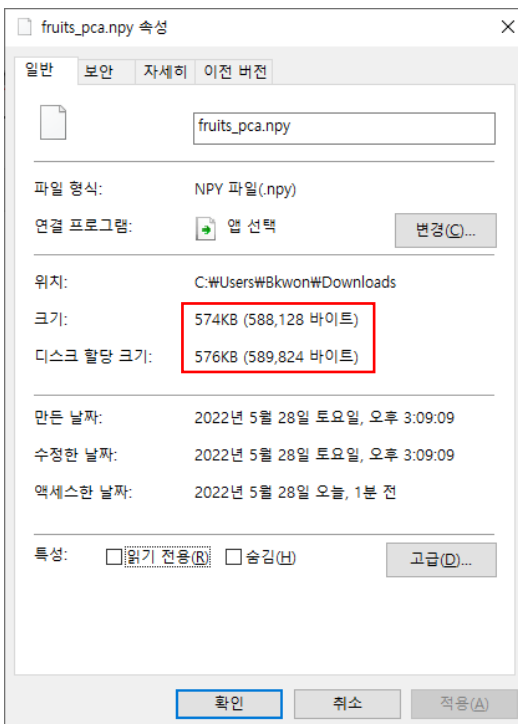
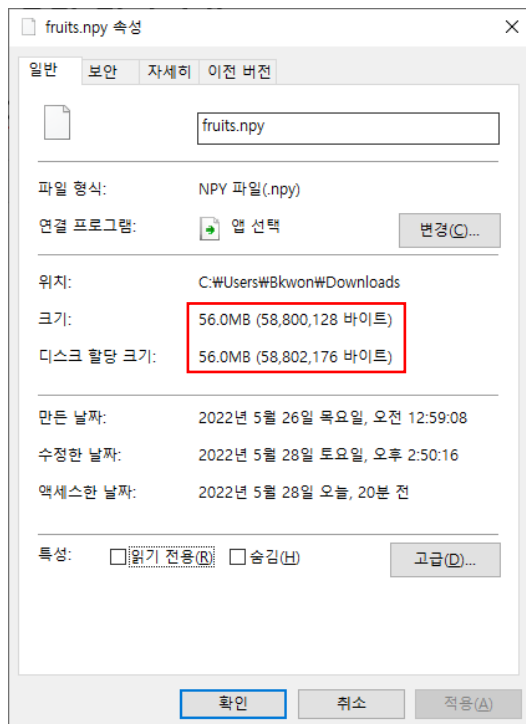
❖ ⑦ 저장 용량 비교하기

- 원본 데이터와 저장 용량을 비교하기 위해서 fruit_pca 배열을 저장해 보자
- 넘파이 배열의 기본 저장 포맷인 npy 파일로 저장하면 됨

```
1 np.save("fruits_pca.npy", fruits_pca)
```

원본 데이터 저장 용량: 56.0MB

PCA 데이터 저장 용량: 574KB



- ✓ 특성의 개수를 10,000개에서 50개로 줄였음
- ✓ fruits_2d 대신 fruits_pca를 저장한다면 저장 공간을 많이 줄일 수 있음

데이터의 차원을 줄였다면
다시 원상 복구할 수도 있을까?

03. PCA 클래스

❖ ⑧ 원본 데이터 재구성

- 앞에서 10,000개의 특성을 50개로 줄였음
- 이로 인해 어느 정도 정보 손실이 발생할 수밖에 없음
- 하지만 최대한 분산이 큰 방향으로 데이터를 투영하였기 때문에 원본 데이터를 상당 부분 재구성할 수 있음
- PCA 클래스는 이를 위해 `inverse_transform()` 메서드를 제공함
- 앞서 50개의 차원으로 축소한 `fruits_pca` 데이터를 전달해 10,000개의 특성으로 복원해 보자

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

실행결과

(1470, 10000)

10,000개의 특성이 복원됨

- `inverse`
(형용사) 역 (반대)의
(형용사) 정반대의

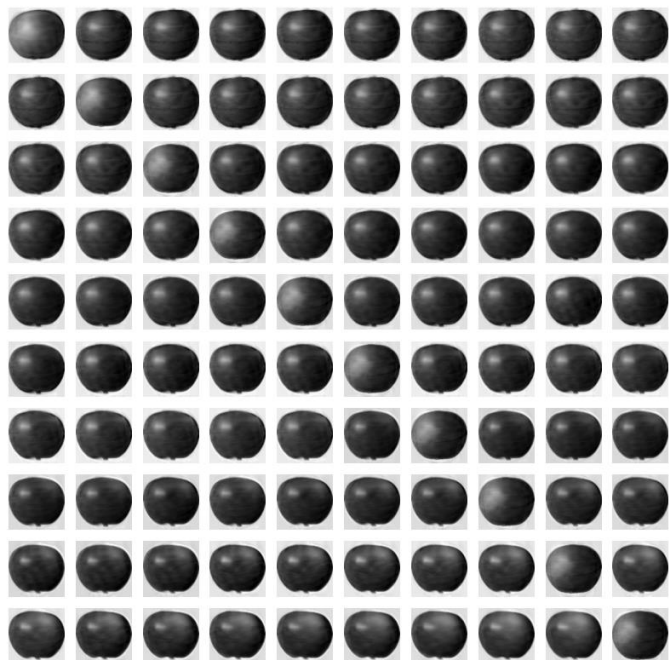
03. PCA 클래스

❖ ⑨ 복원된 데이터를 이미지로 확인하기 (1/3)

- 복원된 데이터를 100×100 크기로 바꾸어 각 과일의 처음 100개의 이미지를 출력해 보겠음
- 이 데이터는 사과, 바나나, 파인애플을 490개씩 담고 있음

```
1 fruits_reconstruct = fruits_inverse.reshape(-1, 100, 100)
2 draw_fruits(fruits_reconstruct[0:100])
```

실행결과



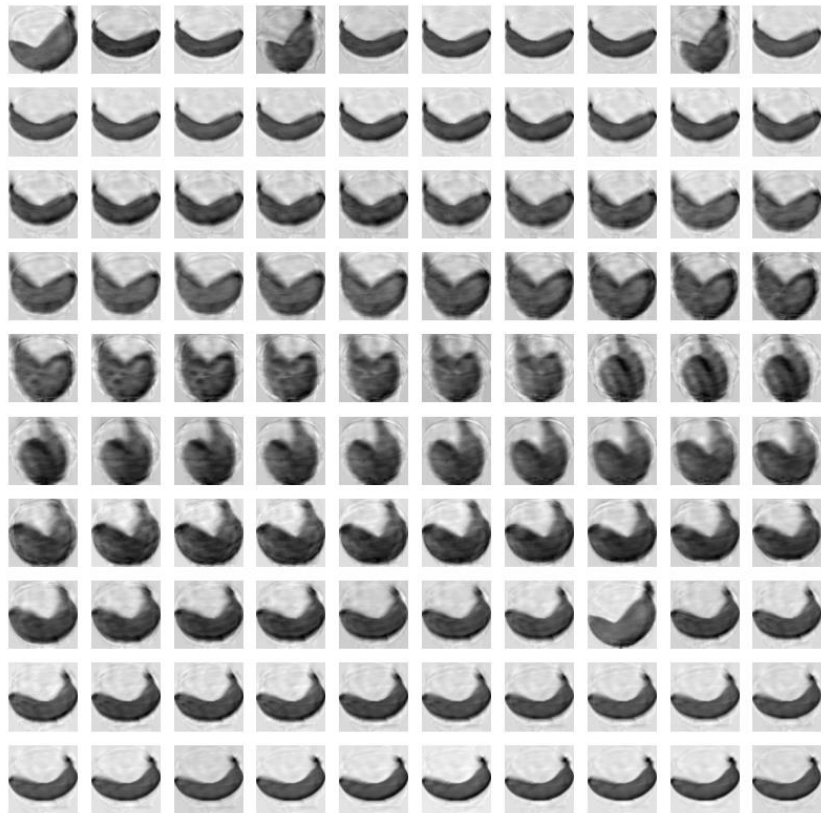
03. PCA 클래스

❖ ⑨ 복원된 데이터를 이미지로 확인하기 (2/3)

- 바나나도 확인해 보자

```
1 draw_fruits(fruits_reconstruct[490:590])
```

실행결과



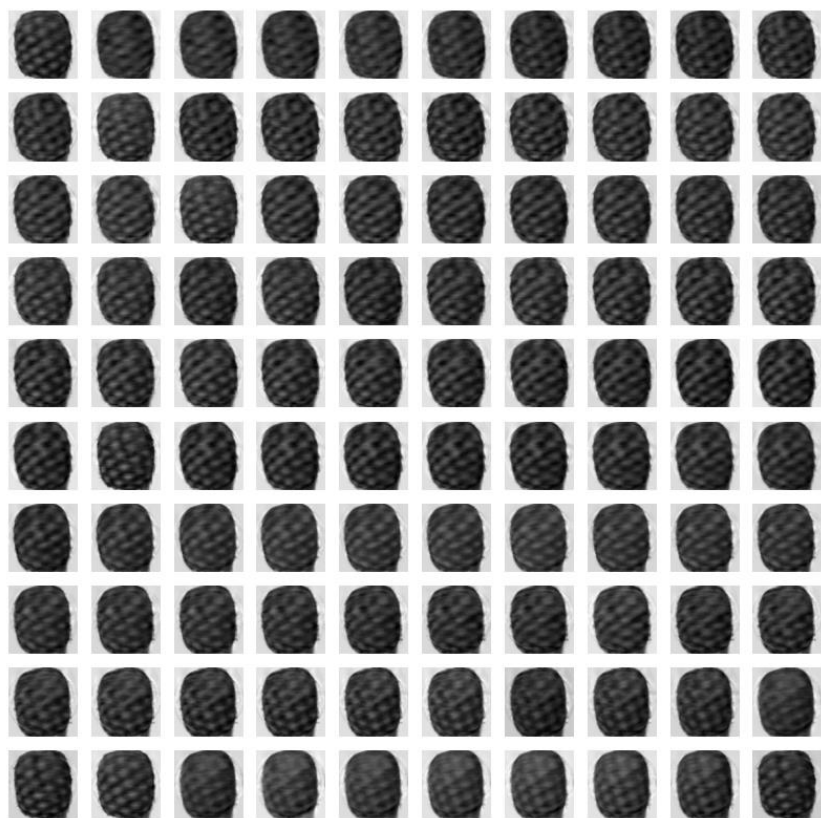
03. PCA 클래스

❖ ⑨ 복원된 데이터를 이미지로 확인하기 (3/3)

- 파인애플도 확인해 보자

```
1 draw_fruits(fruits_reconstruct[980:1080])
```

실행결과



- ✓ 거의 모든 과일이 잘 복원되었음
- ✓ 일부 흐리고 번진 부분이 있지만 불과 50개의 특성을 10,000개로 늘린 것을 감안한다면 놀라운 결과임
- ✓ 이 50개의 특성이 분산을 가장 잘 보존하도록 변환된 것이기 때문임
- ✓ 만약 주성분을 최대로 사용했다면 완벽하게 원본 데이터를 재구성할 수 있을 것임

50개의 특성은 얼마나 분산을 보존하고 있는 것일까?
한 번 알아보도록 하자



03. PCA 클래스

❖ ⑩ 설명된 분산(Explained Variance) (1/3)

- 주성분이 원본 데이터의 분산을 얼마나 잘 나타내는지 기록한 값을 "설명된 분산"이라고 함
- PCA 클래스의 explained_variance_ratio_에 각 주성분의 설명된 분산 비율이 기록되어 있음
- 당연히 첫 번째 주성분의 설명된 분산이 가장 큼

```
1 print(pca.explained_variance_ratio_)
```

실행결과

```
[0.5586464  0.11552605 0.05540004 0.04191264 0.01991335 0.01769261 0.01411554
 0.0112748  0.00895842 0.00870518 0.00713342 0.00686132 0.00618506 0.00535607
 0.00464428 0.00400779 0.00370881 0.00350107 0.00305945 0.00289259 0.00268067
 0.00253049 0.00224965 0.00211854 0.00204749 0.00195585 0.00189895 0.00181368
 0.00170509 0.00161984 0.00161301 0.00150683 0.00142023 0.00138601 0.00133872
 0.00129986 0.00120128 0.00119243 0.00113058 0.0011171  0.00106354 0.00104263
 0.00100836 0.00097745 0.00095493 0.00090986 0.00088153 0.00087676 0.00084975
 0.0008253  ]
```

- ✓ 94%가 넘는 분산을 유지하고 있음
- ✓ 앞에서 50개의 특성에서 원본 데이터를 복원했을 때
원본 이미지의 품질이 높았던 이유를 여기에서 찾을 수 있음

03. PCA 클래스

❖ ⑩ 설명된 분산(Explained Variance) (2/3)

- 이 분산 비율을 모두 더하면 50개의 주성분으로 표현하고 있는 총 분산 비율을 얻을 수 있음

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

실행결과

```
0.9427112901828142
```

설명된 분산의 비율을 그래프로 그려보면
적절한 주성분의 개수를 찾는 데 도움이 됨

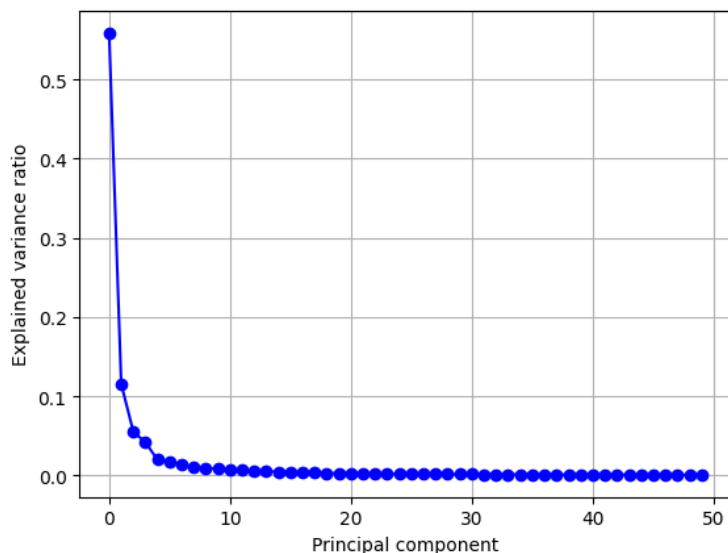
03. PCA 클래스

❖ ⑩ 설명된 분산(Explained Variance) (3/3)

- matplotlib의 plot() 함수로 설명된 분산의 비율을 그래프로 출력해 보겠음

```
1 plt.figure()
2 plt.plot(pca.explained_variance_ratio_, '-o', color='b')
3 plt.xlabel("Principal component")
4 plt.ylabel("Explained variance ratio")
5 plt.grid(True)
6 plt.show()
```

실행결과



- ✓ 그래프를 보면 대략적으로 처음 10개의 주성분이 대부분의 분산을 표현하고 있음
- ✓ 그 다음부터는 각 주성분이 설명하고 있는 분산은 비교적 작음

04. 다른 알고리즘과 함께 사용하기

- 01. 차원과 차원 축소
- 02. PCA 소개
- 03. PCA 클래스
- 05. PCA로 차원 축소
- 06. 마무리

04. 다른 알고리즘과 함께 사용하기

❖ ① 설명된 분산의 비율 지정하여 모델 만들기 (1/2)

- 앞서 PCA 클래스를 사용할 때 n_components 매개변수에 주성분의 개수를 지정했음
- 이 대신 원하는 설명된 분산의 비율을 입력할 수도 있음
- PCA 클래스는 지정된 비율에 도달할 때까지 자동으로 주성분을 찾음

```
1 pca = PCA(n_components=0.6)
2 pca.fit(fruits_2d)
```

실행결과

PCA
PCA(n_components=0.6)

- ✓ 설명된 분산의 60%에 달하는 주성분을 찾도록 PCA 모델을 만들었음
- ✓ 주성분 개수 대신 0~1 사이의 비율을 실수로 입력하면 됨

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

실행결과

2

단 2개의 특성만으로 원본 데이터에 있는
분산의 60%를 표현할 수 있음

04. 다른 알고리즘과 함께 사용하기

❖ ① 설명된 분산의 비율 지정하여 모델 만들기 (2/2)

- 이 모델로 원본 데이터를 변환하겠음
- 주성분이 2개이므로 변환된 데이터의 크기는 (1470, 2)개가 됨

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

실행결과

```
(1470, 2)
```

04. 다른 알고리즘과 함께 사용하기

❖ ② k-Means Clustering Algorithm으로 클러스터 찾기 (1/4)

- 차원 축소된 데이터를 사용해 k-Means Clustering Algorithm으로 클러스터를 찾아보자

```
1 from sklearn.cluster import KMeans
3
4 km = KMeans(n_clusters=3, init="random", random_state=42)
5 km.fit(fruits_pca)
6
7 print(np.unique(km.labels_, return_counts=True))
```

실행결과

```
(array([0, 1, 2]), array([490, 727, 253], dtype=int64))
```

- ✓ fruits_pca로 찾은 클러스터는 각각 490, 727, 253개의 샘플을 포함하고 있음
- ✓ 이는 이전 수업에서 원본 데이터를 사용했을 때와 거의 비슷한 결과임
- ✓ KMeans가 찾은 레이블을 사용해 과일 이미지를 출력해 보자

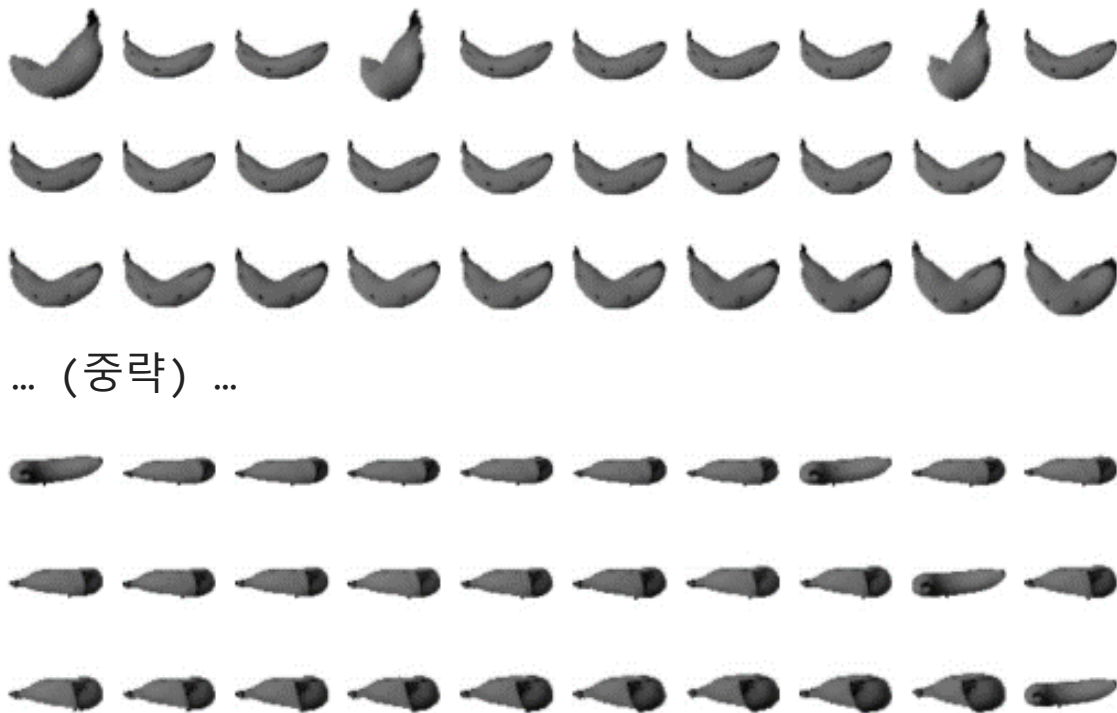
04. 다른 알고리즘과 함께 사용하기

❖ ② k-Means Clustering Algorithm으로 클러스터 찾기 (2/4)

- draw_fruits() 함수를 사용해 레이블이 0인 과일 사진을 모두 그려보자

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

실행결과



레이블이 0인 클러스터는
바나나로만 이루어져 있음

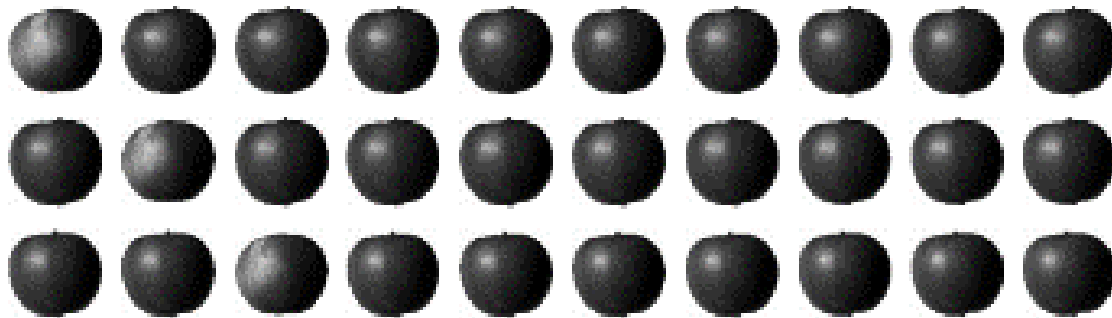
04. 다른 알고리즘과 함께 사용하기

❖ ② k-Means Clustering Algorithm으로 클러스터 찾기 (3/4)

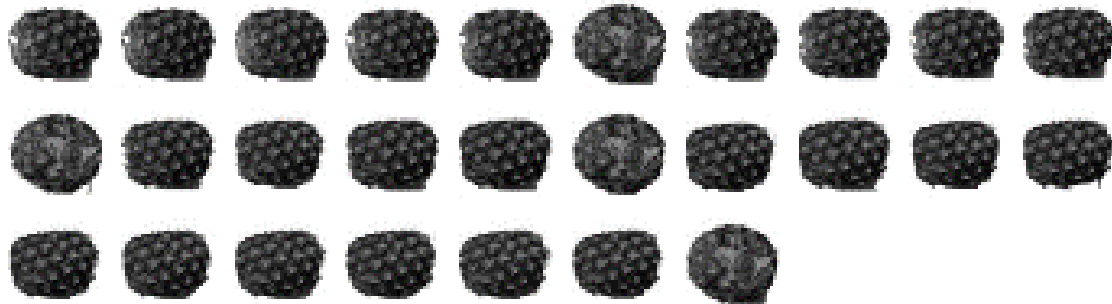
- draw_fruits() 함수를 사용해 레이블이 1인 과일 사진을 모두 그려보자

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

실행결과



... (중략) ...



- ✓ 이전 수업에서 찾은 클러스터와 비슷하게 파인애플은 사과와 조금 혼돈되는 면이 있음
- ✓ 몇 개의 파인애플이 사과 클러스터에 섞여 들어가 있음

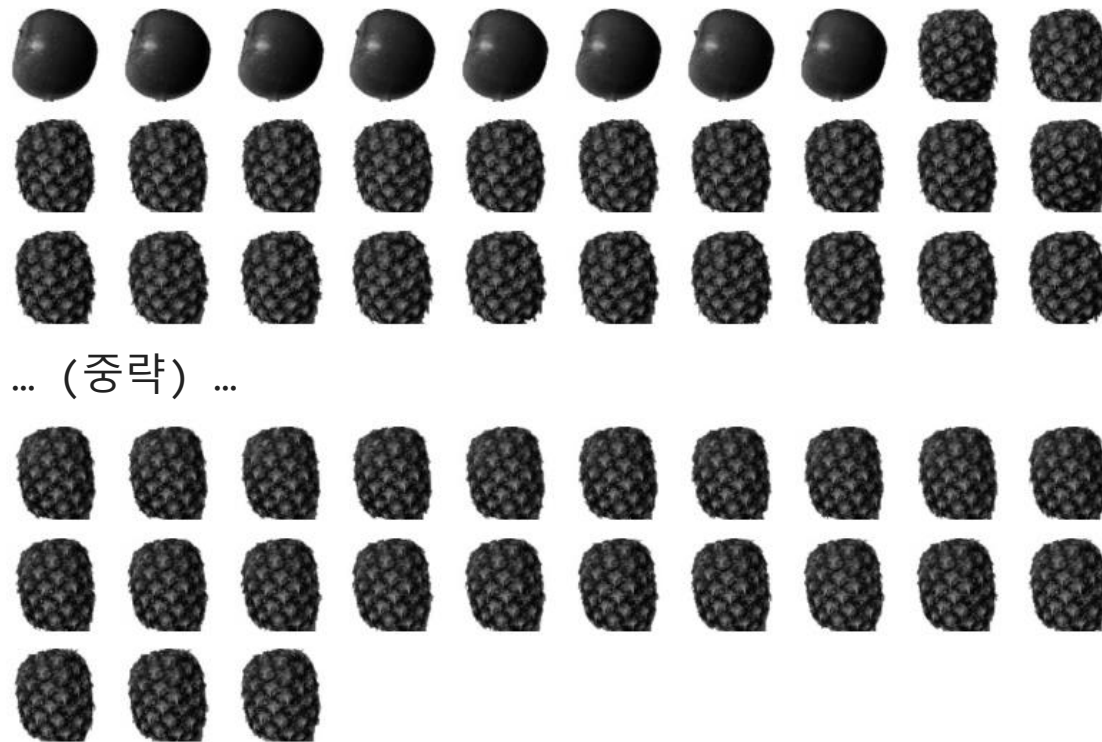
04. 다른 알고리즘과 함께 사용하기

❖ ② k-Means Clustering Algorithm으로 클러스터 찾기 (4/4)

- draw_fruits() 함수를 사용해 레이블이 2인 과일 사진을 모두 그려보자

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

실행결과



몇 개의 사과가 파인애플 클러스터에
섞여 들어가 있음

04. 다른 알고리즘과 함께 사용하기

❖ ③ 산점도로 시각화하기 (1/6)

- 학습 데이터의 차원을 줄이면 또 하나 얻을 수 있는 장점은 시각화임
- 3개 이하로 차원을 줄이면 화면에 출력하기가 비교적 쉬움
- fruits_pca 데이터는 2개의 특성이 있기 때문에 2차원으로 표현할 수 있음

04. 다른 알고리즘과 함께 사용하기

❖ ③ 산점도로 시각화하기 (2/6)

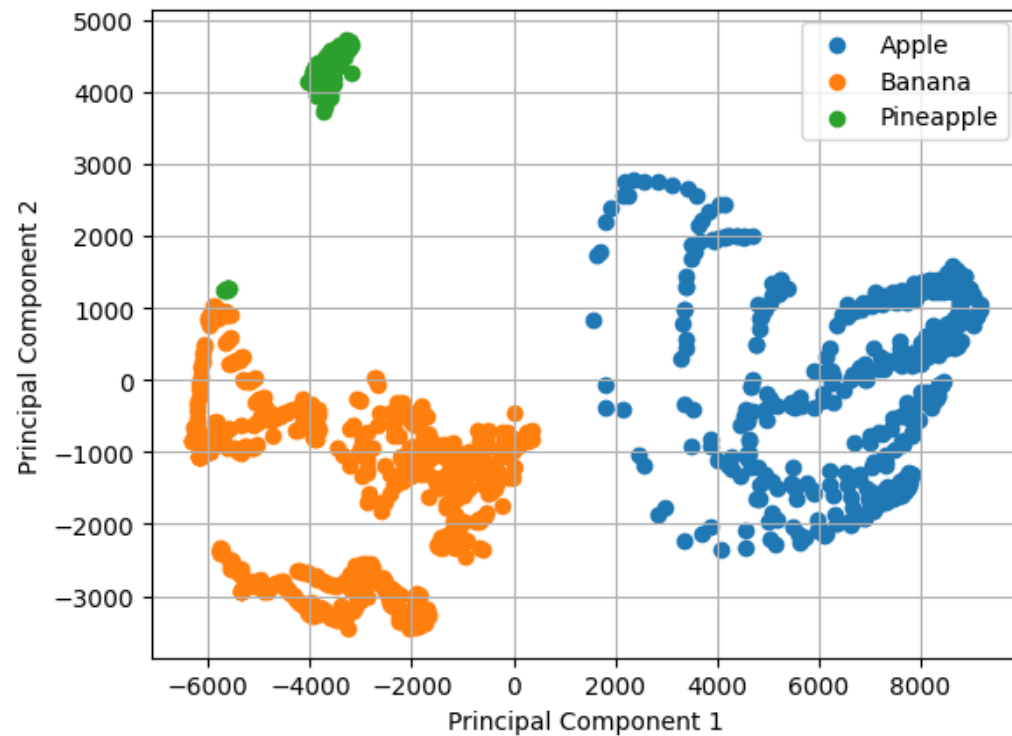
- 앞에서 찾은 km.labels_를 사용해 클러스터별로 나누어 산점도를 그려보자

```
1 plt.figure()
2 for label in range(0, 3):
3     data = fruits_pca[km.labels_==label]
4     plt.scatter(data[:,0], data[:, 1])
5
6 plt.xlabel("Principal Component 1")
7 plt.ylabel("Principal Component 2")
8 plt.legend(["Apple", "Banana", "Pineapple"])
9 plt.grid(True)
10 plt.show()
```


04. 다른 알고리즘과 함께 사용하기

❖ ③ 산점도로 시각화하기 (3/6)

실행결과



- ✓ 각 클러스터의 산점도가 아주 잘 구분되고 있음
- ✓ 각 과일들의 실제 산점도와 비교해 보면 좋을 것 같음

04. 다른 알고리즘과 함께 사용하기

❖ ③ 산점도로 시각화하기 (4/6)

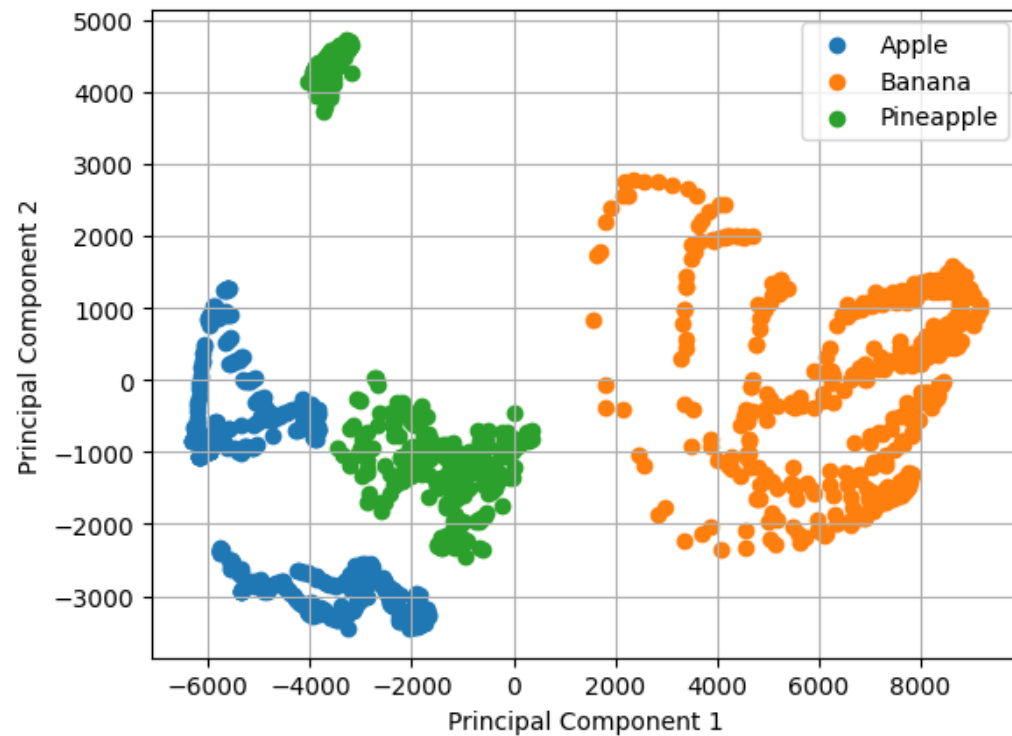
- 각 과일들의 실제 산점도를 그려보자

```
1 plt.figure()
2 plt.scatter(fruits_pca[:490,0], fruits_pca[:490,1])
3 plt.scatter(fruits_pca[490:980,0], fruits_pca[490:980,1])
4 plt.scatter(fruits_pca[980:,0], fruits_pca[980:,1])
5 plt.xlabel("Principal Component 1")
6 plt.ylabel("Principal Component 2")
7 plt.legend(["Apple", "Banana", "Pineapple"])
8 plt.grid(True)
9 plt.show()
```

04. 다른 알고리즘과 함께 사용하기

❖ ③ 산점도로 시각화하기 (5/6)

실행결과

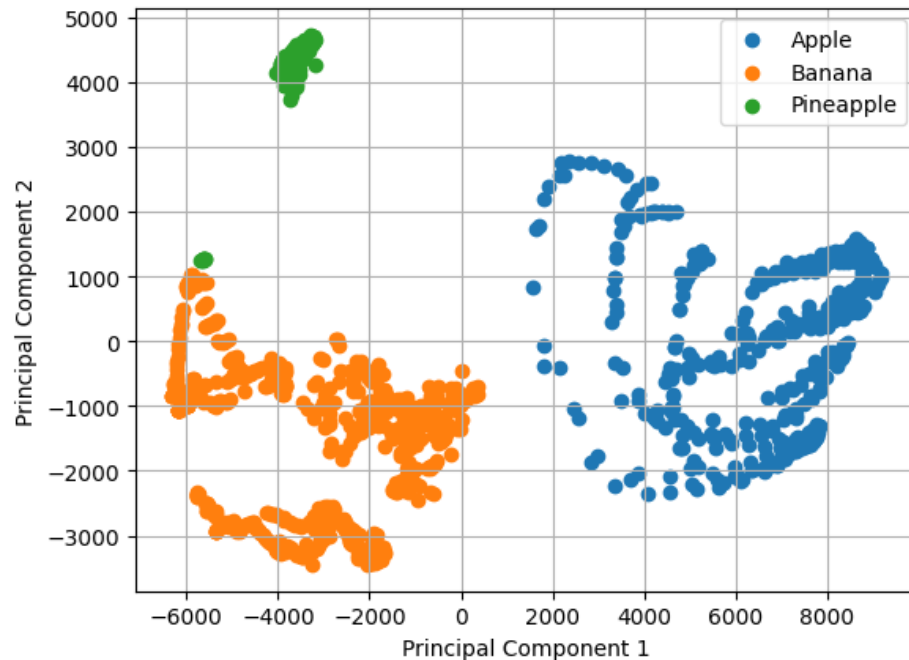


- ✓ 이 그림을 보면 사과와 파인애플 클러스터의 경계가 가깝게 붙어 있음
- ✓ 이 두 클러스터의 샘플은 혼동을 일으키기 쉬울 것 같음

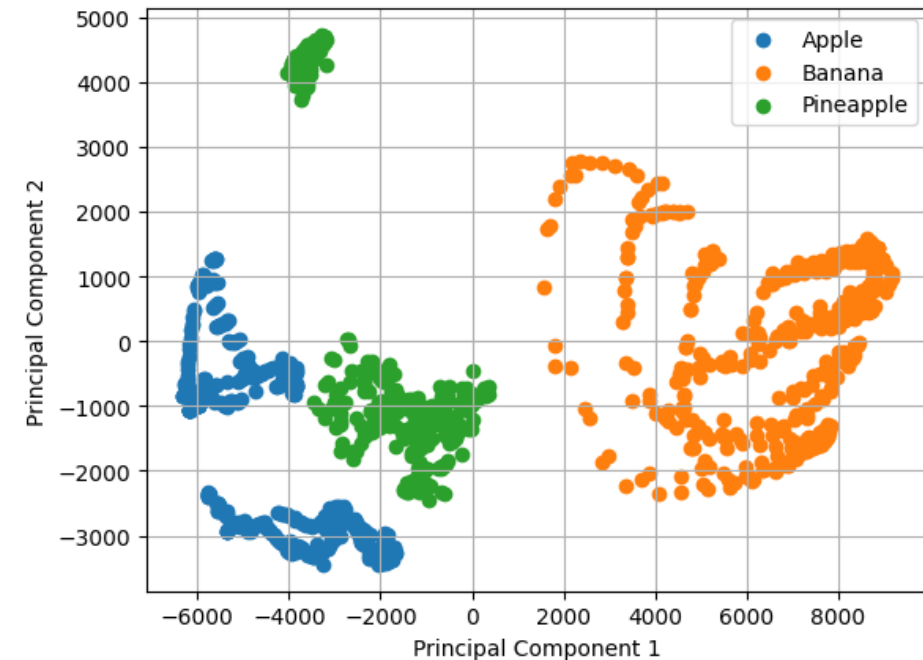
04. 다른 알고리즘과 함께 사용하기

❖ ③ 산점도로 시각화하기 (6/6)

k-Means Clustering Algorithm의 군집 결과



실제 Label



데이터를 시각화하면 예상치 못한 통찰을 얻을 수 있음.
그런 면에서 차원 축소는 매우 유용한 도구 중 하나임



05. 주성분 분석으로 차원 축소

- 01. 차원과 차원 축소
- 02. PCA 소개
- 03. PCA 클래스
- 04. 다른 알고리즘과 함께 사용하기
- 06. 마무리

05. 주성분 분석으로 차원 축소

❖ 전체 과정 요약 (1/2)

- 이번 수업에서는 비지도 학습 문제 중 하나인 차원 축소에 대해 알아보았음
- 차원 축소를 사용하면 데이터셋의 크기를 줄일 수 있고 비교적 시각화하기 쉬움
- 또 차원 축소된 데이터를 지도 학습 알고리즘이나 다른 비지도 학습 알고리즘에 재사용하여 학습 속도를 빠르게 만들 수 있음
- scikit-learn의 PCA 클래스를 사용해 과일 사진 데이터의 특성을 50개로 크게 줄였음
- 특성 개수는 작지만 변환된 데이터는 원본 데이터에 있는 분산의 90% 이상을 표현함
- 이를 설명된 분산(Explained Variance)이라 부름

05. 주성분 분석으로 차원 축소

❖ 전체 과정 요약 (2/2)

- PCA 클래스는 자동으로 설명된 분산을 계산하여 제공해 줌
- 또한 주성분의 개수를 명시적으로 지정하는 대신 설명된 분산의 비율을 설정하여 원하는 비율만큼 주성분을 찾을 수 있음
- PCA 클래스는 변환된 데이터에서 원본 데이터를 복원하는 메서드도 제공함
- 변환된 데이터가 원본 데이터의 분산을 모두 유지하고 있지 않다면 완벽하게 복원되지 않음
- 하지만 적은 특성으로도 상당 부분의 디테일을 복원할 수 있음

지금까지 머신러닝의 주요 알고리즘들을 살펴보았음



05. 주성분 분석으로 차원 축소

❖ 전체 소스 코드 (1/4)

```
1  # 03. PCA 클래스
2  import os
3  import numpy as np
4  import matplotlib.pyplot as plt
5  import matplotlib.image as mpimg
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         R, G, B = img[:, :, 0], img[:, :, 1], img[:, :, 2]
15         imgGray = 0.299 * R + 0.587 * G + 0.114 * B
16         imgGray = np.array(imgGray, dtype='int')
17         imgGray2 = 255 - imgGray
18         fruit_npy.append(imgGray2)
19  fruit_npy = np.array(fruit_npy)
20  print(fruit_npy.shape)
```

```
21  np.save("fruits.npy", fruit_npy)
22
23  fruits = np.load("fruits.npy")
24
25  print(fruits.shape)
26
27  fruits_2d = fruits.reshape(-1, 100*100)
28  print(fruits_2d.shape)
29
30  from sklearn.decomposition import PCA
31
32  pca = PCA(n_components=50)
33
34  pca.fit(fruits_2d)
35
36  print(pca.components_.shape)
37
38
39
40
```


05. 주성분 분석으로 차원 축소

❖ 전체 소스 코드 (2/4)

```
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 draw_fruits(pca.components_.reshape(-1, 100, 100))
62
63 print(fruits_2d.shape)
64
65 fruits_pca = pca.transform(fruits_2d)
66 print(fruits_pca.shape)
67
68 np.save("fruits_pca.npy", fruits_pca)
69
70 fruits_inverse = pca.inverse_transform(fruits_pca)
71 print(fruits_inverse.shape)
72
73 fruits_reconstruct = fruits_inverse.reshape(-1, 100,
74                                             100)
75 draw_fruits(fruits_reconstruct[0:100])
76
77 draw_fruits(fruits_reconstruct[490:590])
78
79 draw_fruits(fruits_reconstruct[980:1080])
80
```

05. 주성분 분석으로 차원 축소

❖ 전체 소스 코드 (3/4)

```
81 print(pca.explained_variance_ratio_)
82
83 print(np.sum(pca.explained_variance_ratio_))
84
85 plt.figure()
86 plt.plot(pca.explained_variance_ratio_, '-o', color='b')
87 plt.xlabel("Principal component")
88 plt.ylabel("Explained variance ratio")
89 plt.grid(True)
90 plt.show()
91
92 # 04. 다른 알고리즘과 함께 사용하기
93 pca = PCA(n_components=0.6)
94 pca.fit(fruits_2d)
95
96 print(pca.n_components_)
97
98 fruits_pca = pca.transform(fruits_2d)
99 print(fruits_pca.shape)
100
```

```
101 from sklearn.cluster import KMeans
102
103 km = KMeans(n_clusters=3, random_state=42)
104 km.fit(fruits_pca)
105 print(np.unique(km.labels_, return_counts=True))
106
107 draw_fruits(fruits[km.labels_==0])
108 draw_fruits(fruits[km.labels_==1])
109 draw_fruits(fruits[km.labels_==2])
110
111 plt.figure()
112 for label in range(0, 3):
113     data = fruits_pca[km.labels_==label]
114     plt.scatter(data[:,0], data[:, 1])
115
116 plt.xlabel("Principal Component 1")
117 plt.ylabel("Principal Component 2")
118 plt.legend(["Apple", "Banana", "Pineapple"])
119 plt.grid(True)
120 plt.show()
```

05. 주성분 분석으로 차원 축소

❖ 전체 소스 코드 (4/4)

```
121 plt.figure()
122 plt.scatter(fruits_pca[:490,0], fruits_pca[:490,1])
123 plt.scatter(fruits_pca[490:980,0], fruits_pca[490:980,1])
124 plt.scatter(fruits_pca[980:,:0], fruits_pca[980:,:1])
125 plt.xlabel("Principal Component 1")
126 plt.ylabel("Principal Component 2")
127 plt.legend(["Apple", "Banana", "Pineapple"])
128 plt.grid(True)
129 plt.show()
```

06. 마무리

- 01. 차원과 차원 축소
- 02. PCA 소개
- 03. PCA 클래스
- 04. 다른 알고리즘과 함께 사용하기
- 05. PCA로 차원 축소

06. 마무리

❖ 키워드로 끝내는 핵심 포인트 (1/2)

차원 축소(Dimensionality Reduction)

- ✓ 원본 데이터의 특성을 적은 수의 새로운 특성으로 변환하는 비지도 학습의 한 종류임
- ✓ 차원 축소는 저장 공간을 줄이고 시각화하기 쉬움
- ✓ 또한 다른 알고리즘의 성능을 높일 수도 있음(성능이 떨어질 수도 있음)

Principal Component Analysis(PCA, 주성분 분석)

- ✓ 차원 축소 알고리즘의 하나로 데이터에서 가장 분산이 큰 방향을 찾는 방법임
- ✓ 이런 방향을 주성분(Principal Component)라고 부름
- ✓ 원본 데이터를 주성분에 투영하여 새로운 특성을 만들 수 있음
- ✓ 일반적으로 주성분은 원본 데이터에 있는 특성 개수보다 작음

06. 마무리

❖ 키워드로 끝내는 핵심 포인트 (2/2)

설명된 분산(Explained Variance)

- ✓ PCA에서 주성분이 얼마나 원본 데이터의 분산을 잘 나타내는지 기록한 것임
- ✓ scikit-learn의 PCA 클래스는 주성분 개수나 설명된 분산의 비율을 지정하여 PCA를 수행할 수 있음

06. 마무리

❖ 핵심 패키지와 함수: **scikit-learn**

PCA

- ✓ PCA를 수행하는 클래스임
- ✓ `inverse_transform()` 메서드는 `transform()` 메서드로 차원을 축소시킨 데이터를 원본 차원으로 복원함
- ✓ 자주 사용하는 매개변수와 속성은 다음과 같음

구분	이름	설명
매개변수 (Parameters)	n_components	✓ 주성분의 개수를 지정함 ✓ 기본값은 None으로 샘플 개수와 특성 개수 중에 작은 것의 값을 사용함
	random_state	✓ 넘파이 난수 시드(Seed) 값을 지정할 수 있음
속성 (Attributes)	components_	✓ 학습 데이터셋에서 찾은 주성분이 저장됨
	explained_variance_	✓ 설명된 분산이 저장됨
	explained_variance_ratio_	✓ 설명된 분산의 비율이 저장됨

06. 마무리

❖ 확인 문제 1.

특성이 20개인 대량의 데이터셋이 있습니다. 이 데이터셋에서 찾을 수 있는 주성분 개수는 몇 개일까요?

= 특성의 개수

- ① 10개
- ② 20개
- ③ 50개
- ④ 100개

06. 마무리

❖ 확인 문제 2.

샘플 개수가 1,000개이고 특성 개수가 100개인 데이터셋이 있습니다.

즉, 이 데이터셋의 크기는 (1000, 100)입니다. 이 데이터를 scikit-learn의 PCA 클래스를 사용해 10개의 주성분을 찾아 변환했습니다. 변환된 데이터셋의 크기는 얼마일까요?

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

06. 마무리

❖ 확인 문제 3.

확인 문제 2번에서 설명된 분산이 가장 큰 주성분은 몇 번째인가요?

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

분산이 가장 큰 주성분부터 찾음
→ 분산의 크기 순으로 내림차순으로 정렬되어 있음!

- ❖ 01. 차원과 차원 축소
- ❖ 02. PCA 소개
- ❖ 03. PCA 클래스
- ❖ 04. 다른 알고리즘과 함께 사용하기
- ❖ 05. PCA로 차원 축소
- ❖ 06. 마무리

THANK YOU!

Q & A

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