



학교

세명컴퓨터고등학교

학과

인공지능소프트웨어과

교사

배장훈

Contents



1	나의 첫 머신러닝	6	비지도 학습
2	데이터 다루기	7	딥러닝 시작
3	회귀 알고리즘과 모델 규제	8	이미지를 위한 인공신경망
4	다양한 분류 알고리즘	9	텍스트를 위한 인공신경망
5	트리 알고리즘	10	프로젝트 발표

비지도 학습

KEWORD

A

군집
알고리즘

KEWORD

B

k-평균

KEWORD

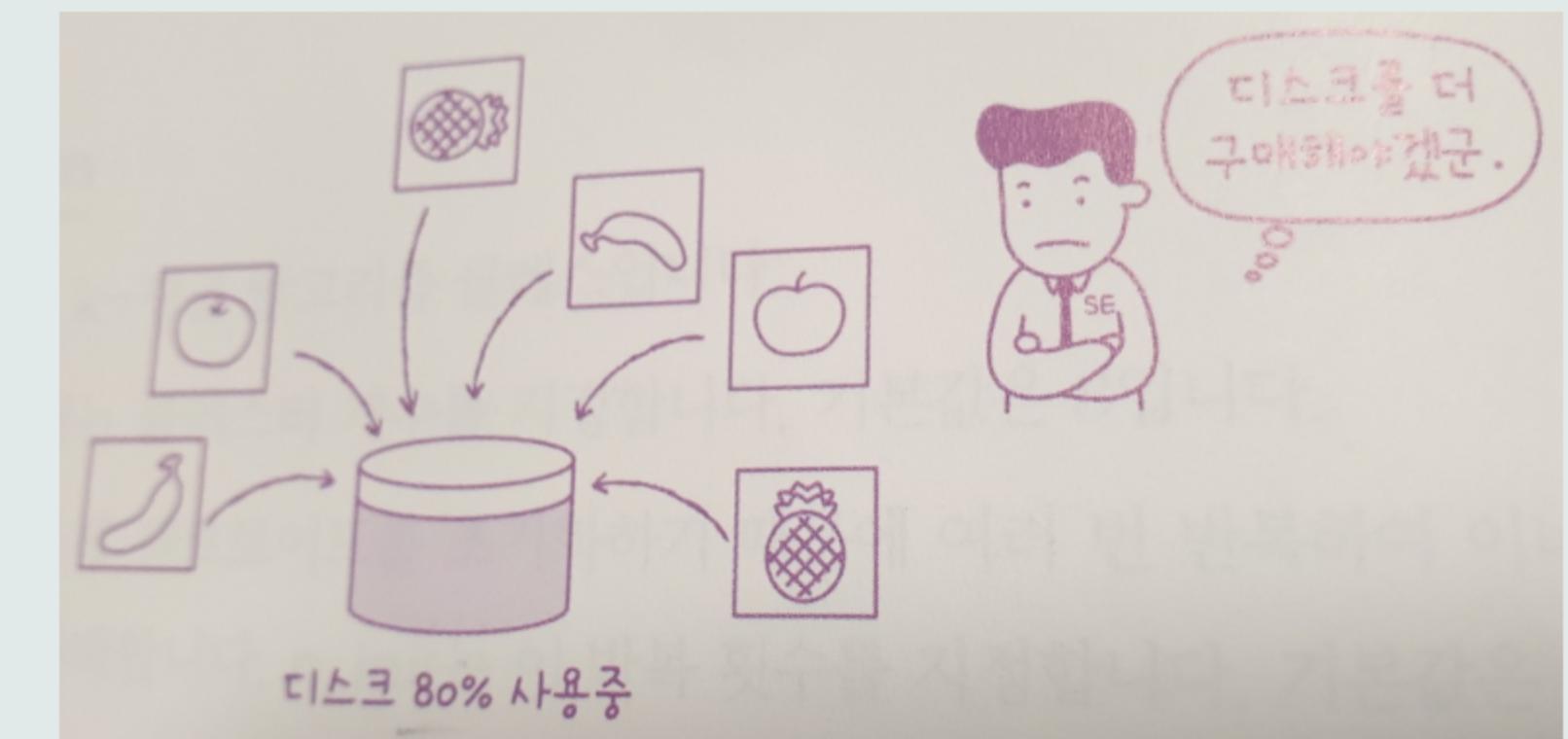
C

주성분
분석

주성분 분석 - 차원 축소

시작하기 전에

1. 매일 각양각색의 과일 사진이 업로드
2. k-평균 알고리즘으로 업로드된 사진을 클러스터로 분류하여 폴더별로 저장
3. 너무 많은 사진이 계속 등록되어 저장 공간이 부족
4. 군집이나 분류에 영향을 끼치지 않으면서
업로드된 사진 용량 줄이는 방법?



차원

특성: 데이터가 가진 속성

예) 과일 사진은 10,000개의 픽셀 → 10,000개의 특성

차원(dimension): 머신러닝에서는 이런 특성을 차원이라고 부름

10,000개의 특성은 결국 10,000개의 차원

차원을 줄일 수 있다면 저장 공간 절약 가능

-> 차원 축소 (dimensionality reduction)

차원 축소

- # 특성이 많으면 선형 모델의 성능이 높아지지만, 훈련 데이터에 과대적합
- # 데이터를 가장 잘 나타내는 일부 특성을 선택하여 데이터 크기를 줄임
- # 지도 학습 모델의 성능 향상
- # 줄어든 차원에서 다시 원본 차원으로 손실 줄이면서 복원 가능

주성분 분석(principal component analysis)

- 대표적 차원 축소 알고리즘, 줄여서 PCA

PCA 손코딩

이전 페이지와 동일한 데이터 사용

과일 사진 데이터 다운로드하여 넘파이 배열로 적재

```
!wget https://bit.ly/fruits\_300\_data -O fruits_300.npy  
import numpy as np  
fruits = np.load('fruits_300.npy')  
fruits_2d = fruits.reshape(-1, 100*100)
```

PCA 손코딩

사이킷런은 주성분 분석 알고리즘 제공

`sklearn.decomposition` 모듈 아래 `PCA` 클래스

`n_components` 매개변수에 주성분 개수 지정

비지도 학습이기 때문에 `fit()` 메서드에 타깃값 제공 X

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

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

PCA 손코딩

PCA 클래스가 찾은 주성분은 **components_** 속성에 저장

```
print(pca.components_.shape)  
  
(50, 10000)
```

50: 50개의 주성분

10,000: 원본 데이터의 특성 개수

원본 데이터와 차원이 같으므로

주성분을 **100 × 100 이미지 출력 가능**

이전 pdf의 **draw_fruits()** 함수 사용

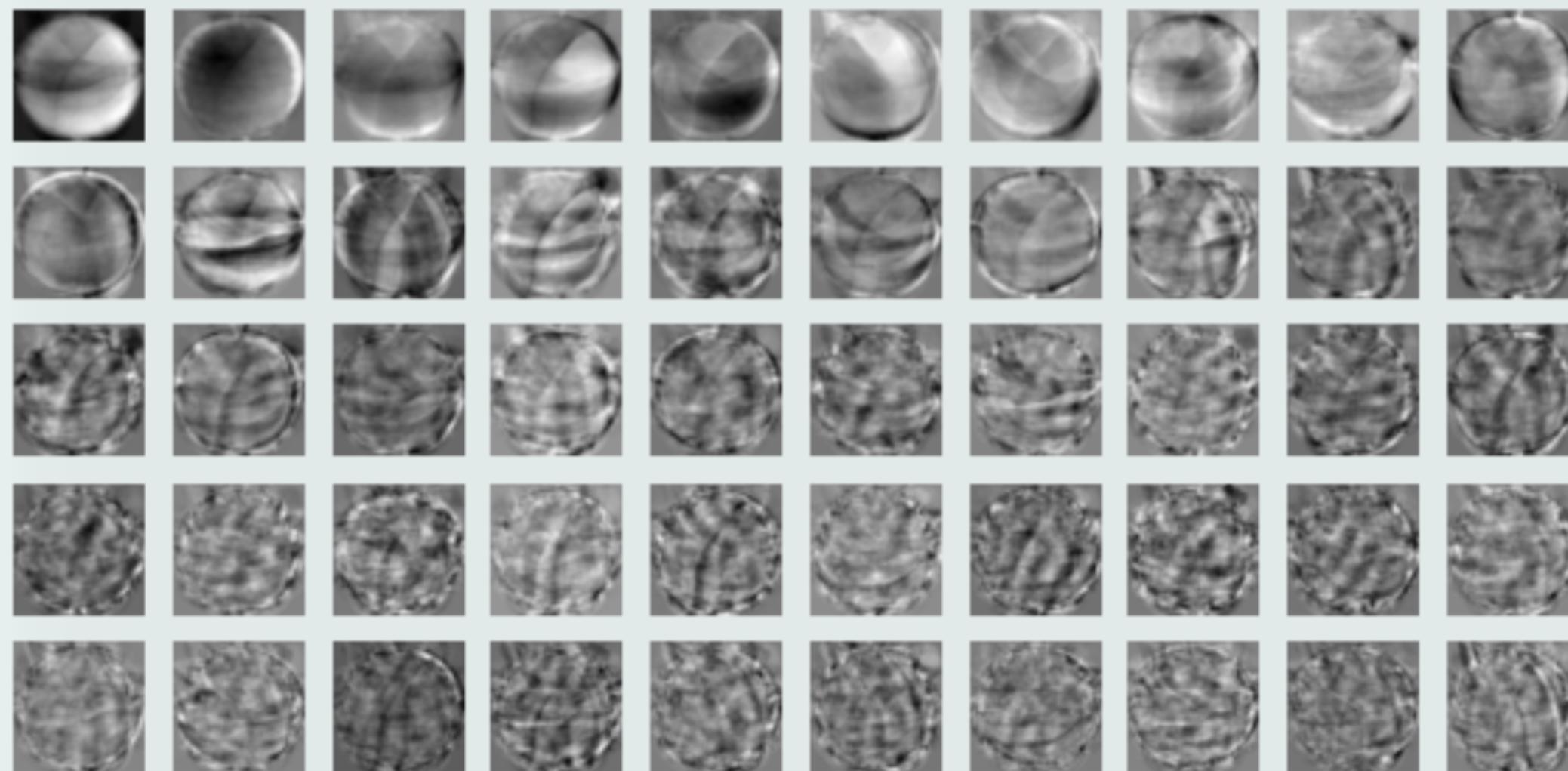
(복사해서 사용)

```
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  
    rows = int(np.ceil(n/10))  
    # 행이 1개 이면 열 개수는 샘플 개수입니다. 그렇지 않으면 10개로  
    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()
```

PCA 손코딩

주성분 50개를 100×100 크기의 이미지로 시각화

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



데이터셋에 있는
어떤 특징을 잡아낸 것

PCA 손코딩

원본 데이터를 주성분에 투영

→ 특성 개수를 10,000개에서 50개로 줄임

원본 데이터를 각 주성분으로 분해

PCA의 `transform()` 메서드 사용해 원본 데이터의 차원 축소

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

(300, 10000)
(300, 50)
```

- 50개의 주성분을 찾은 PCA 모델 사용해 (300, 50) 크기의 배열로 변환
- `fruits_pca` 배열은 50개의 특성을 가진 데이터
- 1/200으로 줄임: 데이터 저장 공간 줄일 수 있음

원본 데이터 재구성

10,000개의 특성을 50개로 줄임 → 어느 정도 손실 발생

inverse_transform()

- 사이킷런은 원본 데이터를 상당 부분 재구성 및 복원 가능
- 50개 차원으로 축소한 fruits_pca 데이터로 다시 10,000개 특성 복원

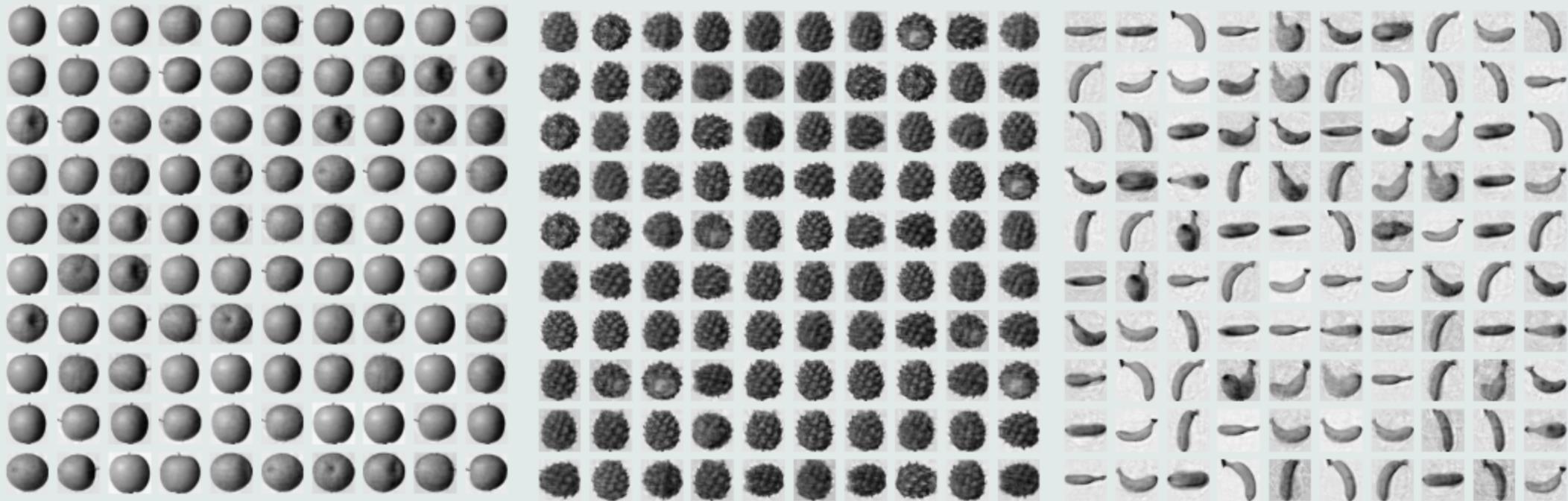
```
fruits_inverse = pca.inverse_transform(fruits_pca)
print(fruits_inverse.shape)

(300, 10000)
```

복원 데이터 시각화

100×100 크기로 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")
```



- 거의 모든 과일이 잘 복원
- 50개의 특성을 10,000개로 늘렸음
- 50개의 특성이 분산을 가장 잘 보존하도록 변환된 것
- 신기방기

로지스틱 회귀 모델

원본 데이터와 PCA로 축소한 데이터를 로지스틱 회귀 모델에 적용해서 비교

지도 학습 모델은 타깃값 필요: 사과 0, 파인애플 1, 바나나 2 지정

```
from sklearn.linear_model import LogisticRegression  
lr = LogisticRegression()  
target = np.array([0] * 100 + [1] * 100 + [2] * 100)
```

원본 데이터 fruits_2d 사용

```
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  
1.6316192626953125
```

- 교차 검증 점수: 0.997 (특성이 10,000개
인데 샘플이 300개이므로 **과대적합 가능**)
- **fit_time**: 각 교차 검증 폴드의 훈련 시간
여기서는 1.63초 걸림

로지스틱 회귀 모델

PCA로 축소한 fruits_pca 사용해서 다시 교차 검증

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

```
1.0  
0.07665228843688965
```

- 교차 검증 점수: 1
50개의 특성만 사용했음에도 정확도 100%
- fit_time: 0.07초
- 저장 공간 분 아니라 훈련 속도도 높일 수 있음

최적의 주성분 개수

기준에는 **n_component** 매개변수에 주성분 개수 50으로 직접 지정

원하는 분산의 비율 입력 가능: 주로 0.5 사용

지정된 비율에 도달할 때까지 자동으로 주성분 찾음

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

```
2
```

- 분산의 50% 달하는 주성분 찾도록 PCA 모델 생성
- 단 2개의 특성!

최적의 주성분 개수

주성분 2개인 모델로 원본 데이터 변환

```
fruits_pca = pca.transform(fruits_2d)
print(fruits_pca.shape)
(300, 2)
```

주성분 2개이므로 변환된 데이터 크기는 (300, 2)

특성 2개로 교차 검증

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

/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_logistic.py:818: ConvergenceWarning: ...
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
https://scikit-learn.org/stable/modules/linear\_model.html#logistic-regression
extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG,
0.9933333333333334
0.08121962547302246
```

- 2개 특성 사용했음에도 99.3 % 정확도
- 0.08초

k-평균 알고리즘

2개로 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([110, 99, 91]))
```

- 각각 91개, 99개, 110 샘플 포함
- 원본 데이터를 사용했을 때와 거의 비슷한 결과

k-평균 알고리즘

2개로 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([110, 99, 91]))
```

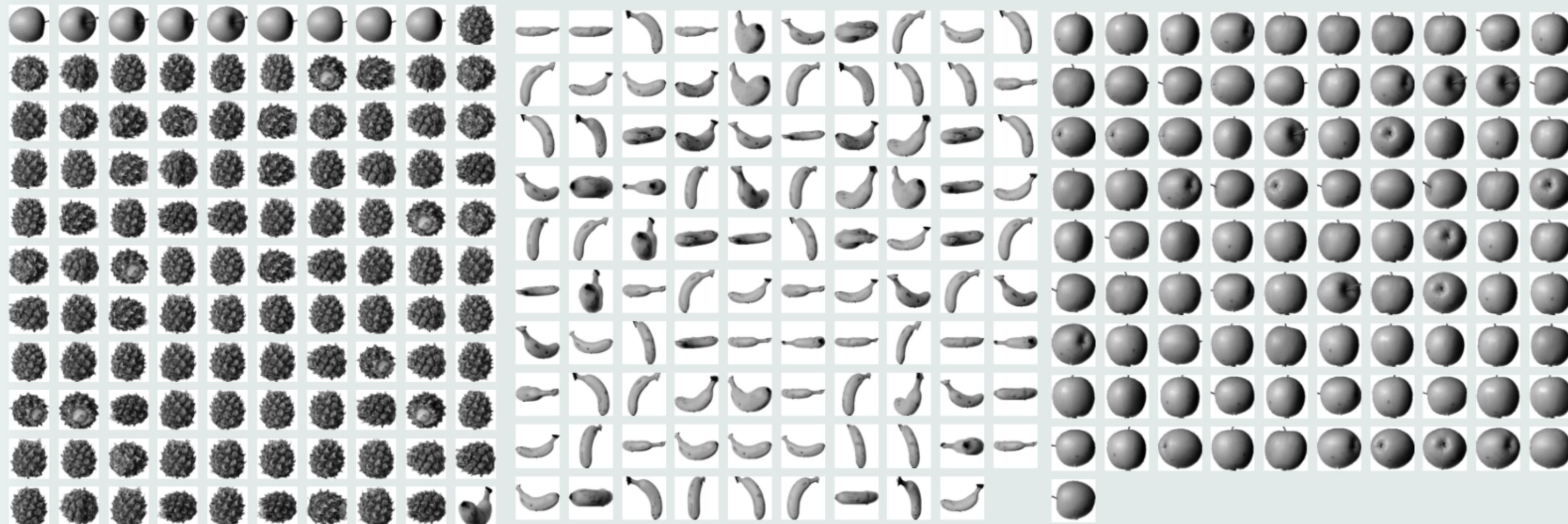
- 각각 91개, 99개, 110 샘플 포함
- 원본 데이터를 사용했을 때와 거의 비슷한 결과

k-평균 알고리즘 시각화

레이블 별로 파일 이미지 시각화

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

- 원본 데이터와 비슷하게 파인애플/사과는 조금 혼돈
- 하지만 훨륭한 결과



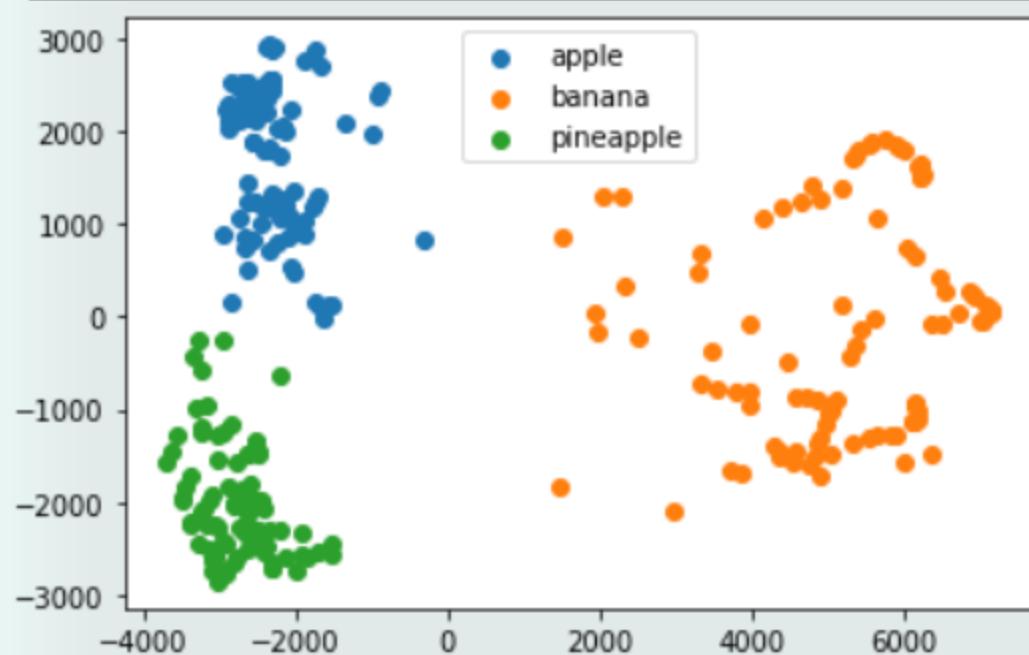
k-평균 알고리즘 시각화

차원 축소의 또 다른 장점은 시각화

3개 이하로 차원 줄이면 화면에 출력하기 비교적 쉬움

fruits_pca 데이터는 2개 특성 → 2차원 표현 가능

```
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()
```



- 각 클러스터의 산점도가 아주 잘 구분됨
- 2개 특성만 사용했는데 성능 뛰어난 이유 보임
- 사과와 바나나 경계가 깊게 뚫어 있어
샘플 몇 개는 혼동

**THANK
YOU**