



HAJ Lecture

Chapter 8. 차원 축소
Chapter 9. 비지도 학습

Hands-On Machine Learning
with Scikit-Learn, Keras & TensorFlow

| 8.1 차원의 저주

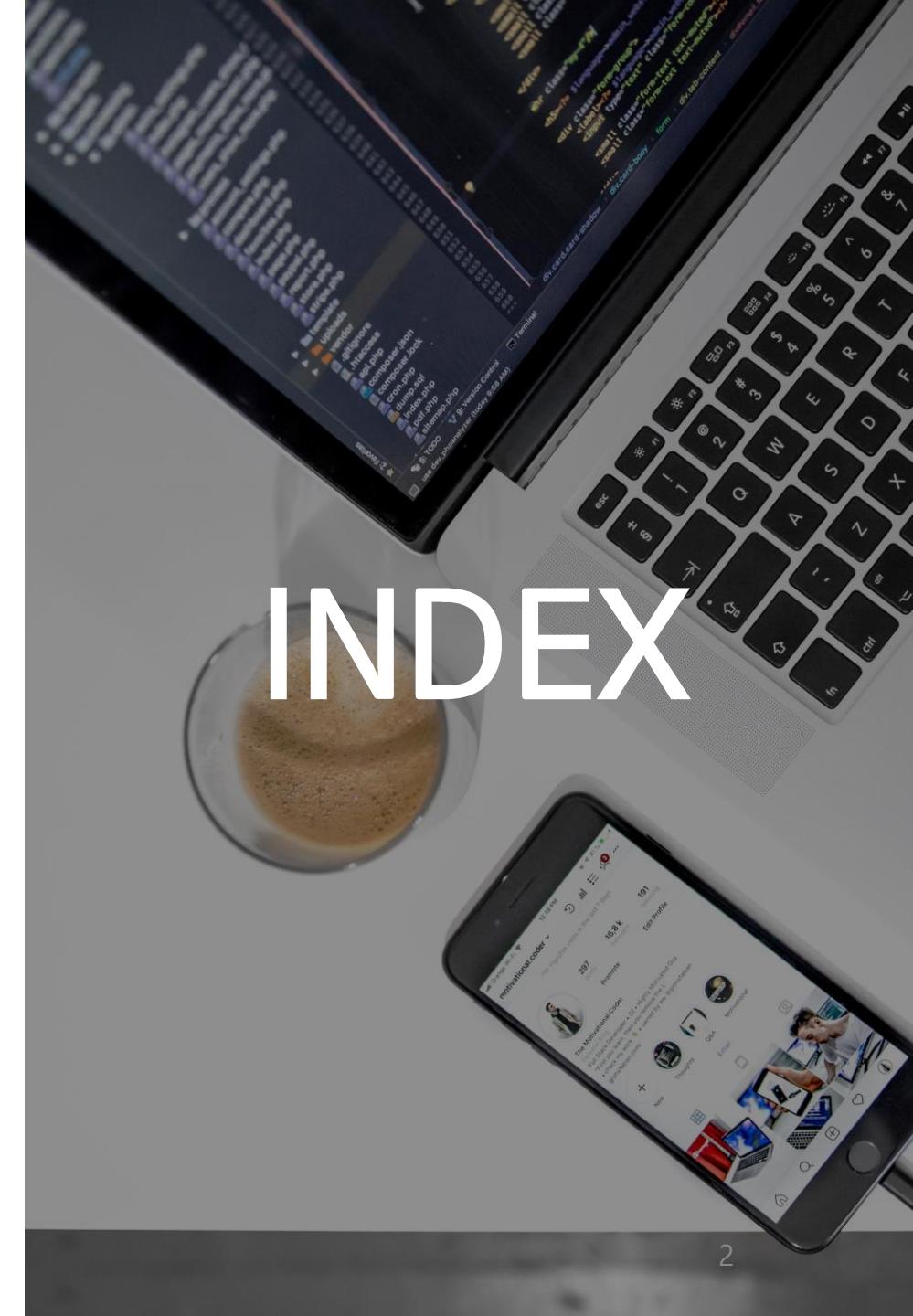
| 8.2 차원 축소를 위한 접근 방법

| 8.3 PCA

| 8.4 커널 PCA

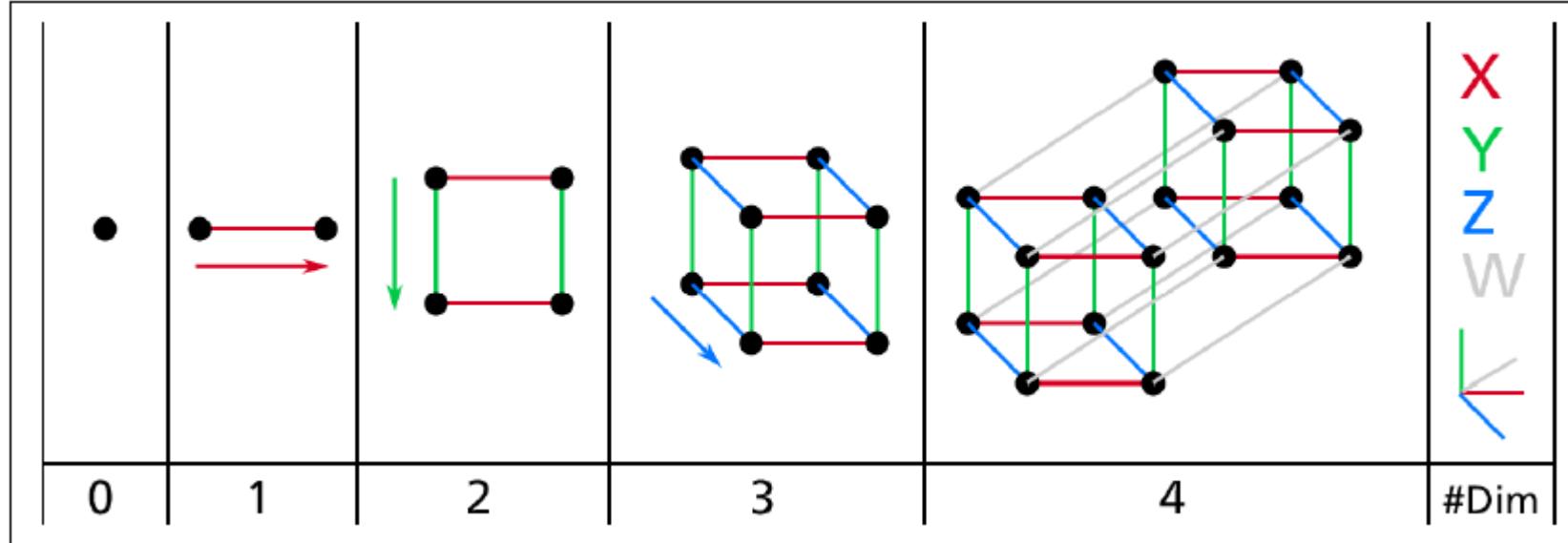
| 8.5 LLE

| 8.6 다른 차원 축소 기법



| 8.1 차원의 저주

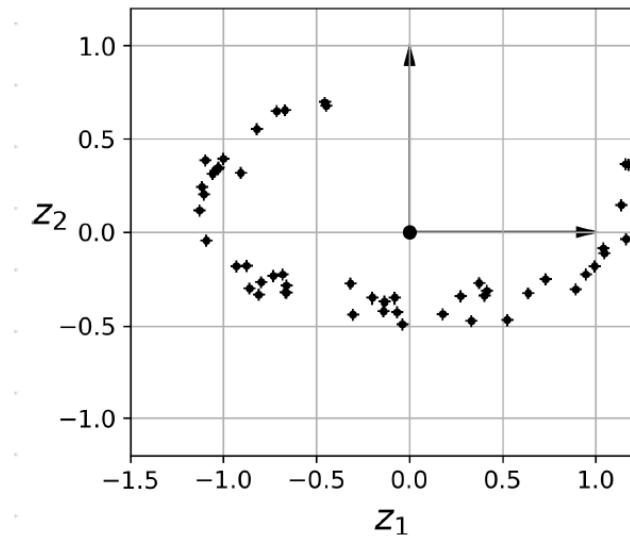
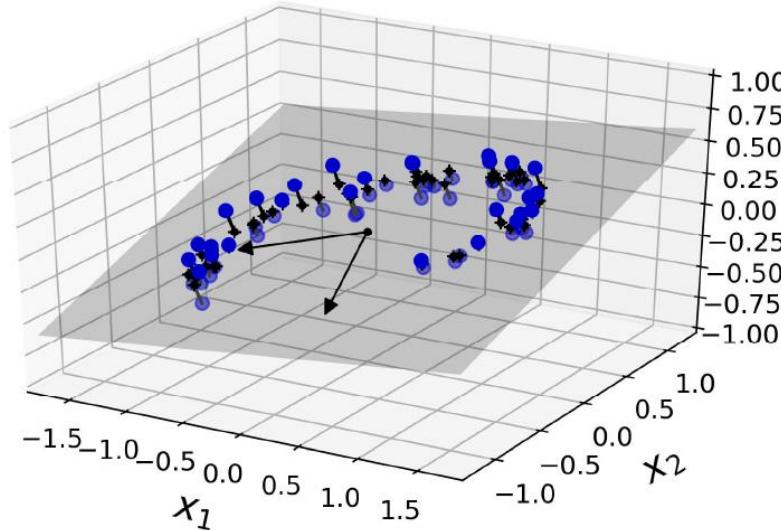
- 훈련 샘플 각각이 매우 많은 수의 feature를 가지고 있는 경우
 - 훈련의 속도를 더디게 만든다.
 - 좋은 솔루션을 찾기 어렵게 만든다.
 - 해결책: feature의 수를 줄여 가능한 범위로 변경할 수 있다.
- 두 가지 접근방법: 투영(projection) / 매니폴드 학습(manifold learning)



| 8.2 차원 축소를 위한 접근 방법

투영(Projection)

대부분의 경우, 훈련샘플이 모든 차원에 걸쳐 균일한 분포를 띠지는 X
→ 고차원 공간 안의 저차원 부분 공간에 놓여 있다.

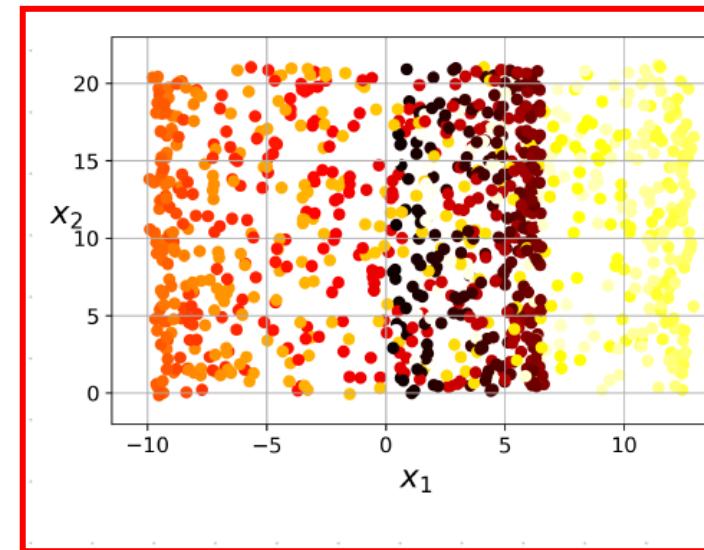
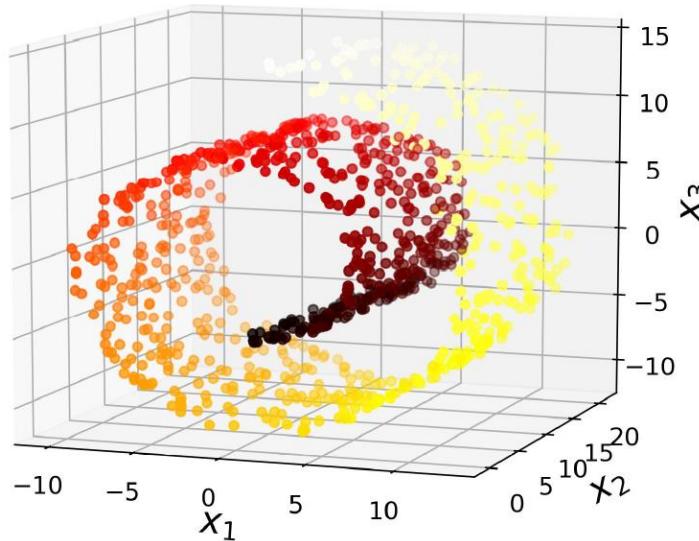


- 데이터셋의 차원을 3D에서 2D로 줄임
- 각 축은 새로운 특성 z_1 과 z_2 에 대응된다.

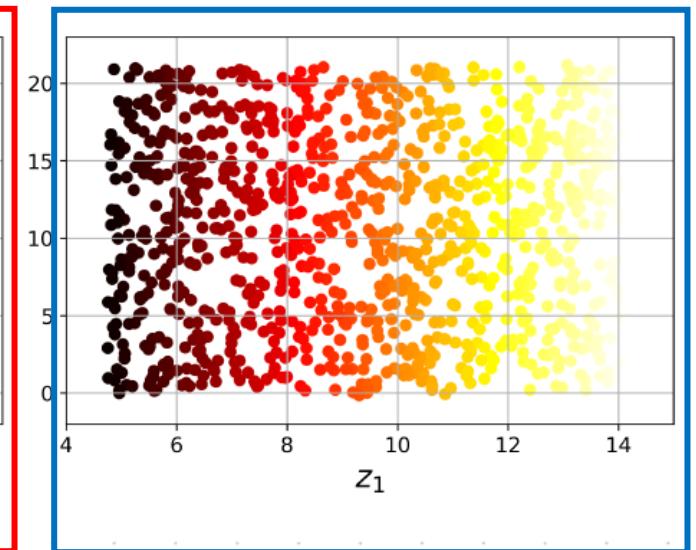
8.2 차원 축소를 위한 접근 방법

투영(Projection)

Q. 데이터가 스위스롤(swiss roll)의 분포를 띠고 있다면? (=부분공간의 뒤틀림)



평면에 투영시킬 경우
→ 스위스 톤의 층이 서로 둉개짐



Desired.
스위스 톤을 펼쳐 2D 데이터셋 얻기

| 8.2 차원 축소를 위한 접근 방법

매니폴드 학습(Manifold Learning)

스위스 롤 → 매니폴드(manifold)의 한 형태

→ 2D 매니폴드: 고차원 공간에서 휘어지거나 뒤틀린 2D 모양

→ D차원 매니폴드: 국부적으로 D차원 초평면으로 보일 수 있는
n차원 공간의 일부($n < D$)

- 매니폴드 학습

훈련샘플이 놓인 매니폴드를 모델링하는 것

- 매니폴드 가정(manifold assumption)

“실제 고차원 데이터셋이 더 낮은 저차원 매니폴드에 가깝게 놓여있다.”라는
가정으로 매니폴드 학습의 기본 가정

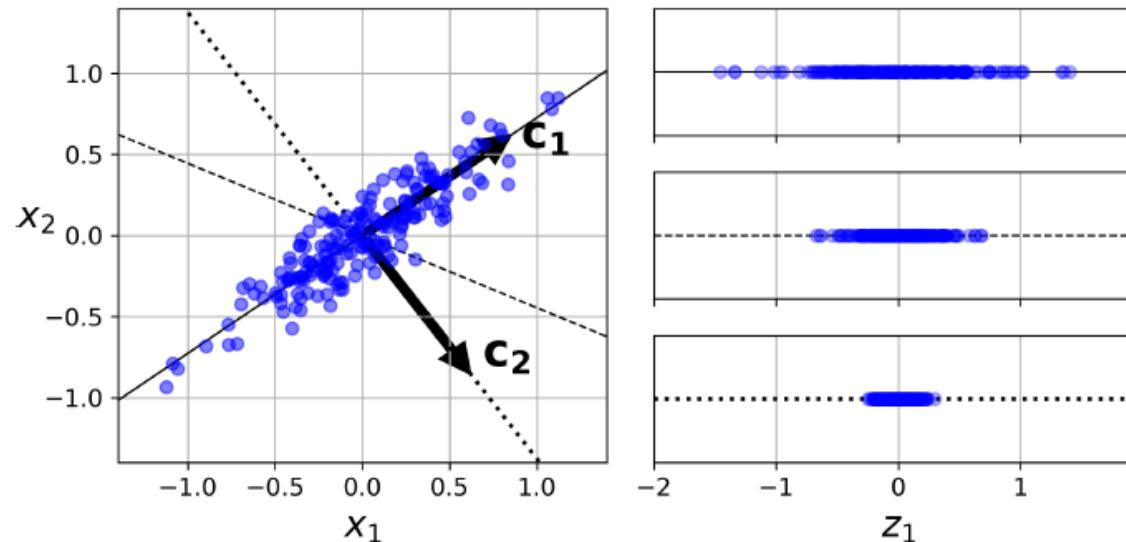
8.3 PCA

주성분 분석(PCA; Principal Component Analysis)

데이터에 가장 가까운 초평면(hyper-plane)을 정의한 후,

데이터를 이 평면에 투영시킴(이것을 단면으로 잘라낸다고 생각하면 쉬움)

- **분산 보존** → 분산이 최대로 보존되는 축을 선택하는 것이
정보가 가장 적게 손실되므로 합리적이다.



분산의 정도가 큼(→최대로 보존)

분산의 정도가 중간

분산의 정도가 작음

8.3 PCA

주성분 분석(PCA; Principal Component Analysis)

- 주성분(principal component; PC)

고차원 데이터셋의 경우 PCA는 서로 직교하는 축을 찾는데

N차원 데이터셋의 경우 n번째 축까지 찾는다.

i번째 축을 이 데이터의 i번째 주성분이라고 지칭한다.

- 주성분을 어떻게 찾는가?

특이값 분해(SVD; singular value decomposition)

: 표준적 행렬 분해 기법 → 훈련세트 행렬 X 를 3개의 행렬의 곱셈으로 표현

8.3 PCA

주성분 분석(PCA; Principal Component Analysis)

- 특이값 분해(SVD; singular value decomposition)
 - : 표준적 행렬 분해 기법 → 훈련세트 행렬 X 를 3개의 행렬의 곱셈으로 표현
- 주성분 행렬(principal components matrix)

$$V = \begin{pmatrix} | & | & \cdots & | \\ c_1 & c_2 & \cdots & c_n \\ | & | & \cdots & | \end{pmatrix}$$

- numpy의 svd() 함수를 이용한 주성분 구하기

```
X_centered = X - X.mean(axis=0)
U, s, Vt = np.linalg.svd(X_centered)
c1 = Vt.T[:, 0]
c2 = Vt.T[:, 1]
```

8.3 PCA

d차원으로 투영하기

- 주성분 추출 → 처음 d개의 주성분으로 정의한 초평면에 투영
→ 데이터셋의 차원을 d차원으로 축소
→ 이렇게 만들어진 초평면은 분산을 최대로 보존하는 투영임이 보장됨
- 훈련세트를 d차원으로 투영하기

$$\mathbf{X}_{d\text{-proj}} = \mathbf{X}\mathbf{W}_d$$

- with Python: 2개의 주성분으로 정의된 평면에 훈련세트 투영하기

```
W2 = Vt.T[:, :2]
```

```
X2D = X_centered.dot(W2)
```

8.3 PCA

Scikit-Learn 사용하기

- Scikit-Learn의 PCA 모델은 SVD 분해를 사용하여 구현됨
- 예시코드: 데이터셋의 차원을 2로 줄이기
 - (여기에서 PCA 모델은 자동으로 데이터를 중앙에 맞춰준다.)
 - components_ 속성에는 W^d 의 transpose가 담겨 있다.

```
from sklearn.decomposition import PCA
```

```
pca = PCA(n_components = 2)  
X2D = pca.fit_transform(X)
```

8.3 PCA

설명된 분산의 비율

`explained_variance_ratio_`: 주성분의 설명된 분산의 비율

→ 이는 각 주성분의 축을 따라 있는 데이터셋의 분산 비율을 나타낸다.

```
>>> pca.explained_variance_ratio_
array([0.84248607, 0.14631839])
```

데이터셋 분산의 84.2%가 첫번째 PC를 따라 놓여있고,
14.6%가 두번째 PC를 따라 놓여있다.
세번째 PC에는 1.2% 미만 → 아주 작은 양

적절한 차원 수 선택하기

차원 축소를 하지 않고 PCA를 계산한 후 훈련세트의 분산을 95%로

유지하는 데 필요한 최소한의 차원 수 계산하기

```
pca = PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
d = np.argmax(cumsum >= 0.95) + 1
```

8.4 커널 PCA

커널 PCA(kPCA; Kernel PCA)

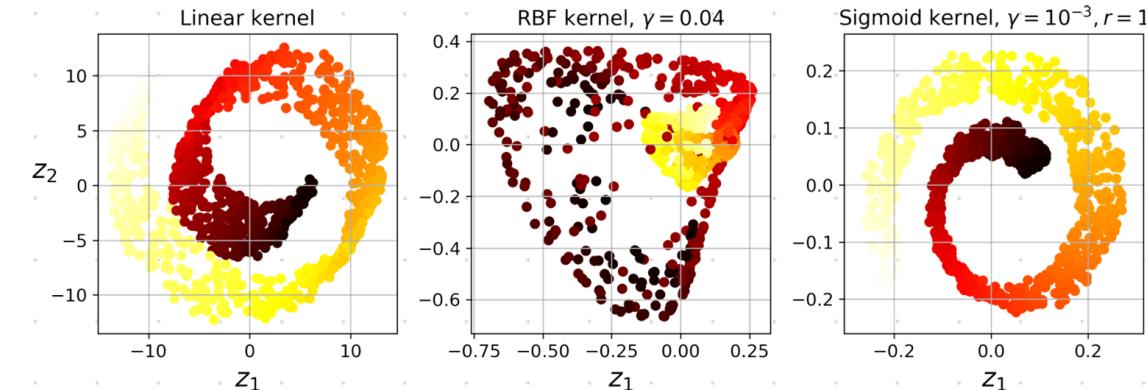
차원 축소를 위해 복잡한 비선형 투영을 수행하는 것

→ 투영된 후에 샘플의 군집을 유지하거나 꼬인 매니폴드에 가까운 데이터셋을 펼칠 때도 유용하다.

in Scikit-Learn, KernelPCA

```
from sklearn.decomposition import KernelPCA
```

```
rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.04)
X_reduced = rbf_pca.fit_transform(X)
```



8.4 커널 PCA

커널 선택과 하이퍼파라미터 튜닝

kPCA : 비지도학습 → 커널/하이퍼파라미터 선택에 있어 명확한 성능기준 X

- 1) 2 stage의 파이프라인을 만든다.
- 2) kPCA를 사용하여 2차원 축소를 하고, 분류를 위한 로지스틱 회귀 적용
- 3) 마지막 stage에서 GridSearchCV를 이용해 kPCA의 가장 좋은 커널과 gamma 파라미터를 찾는다.

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.pipeline import Pipeline
```

(결과)

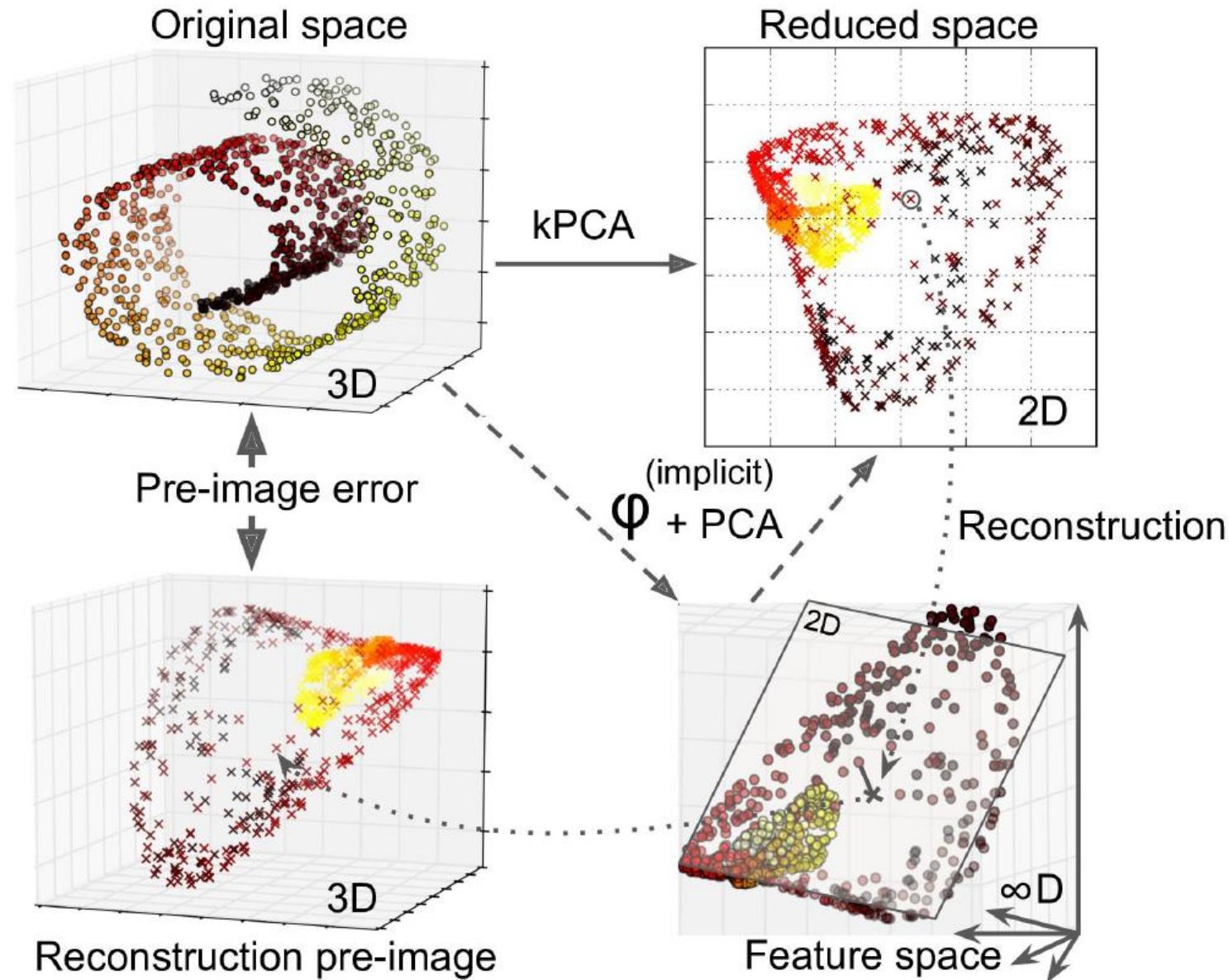
```
>>> print(grid_search.best_params_)
{'k pca_gamma': 0.04333333333333335, 'k pca_kernel': 'rbf'}
```

```
clf = Pipeline([
    ("k pca", KernelPCA(n_components=2)),
    ("log reg", LogisticRegression())
])

param_grid = [
    "k pca_gamma": np.linspace(0.03, 0.05, 10),
    "k pca_kernel": ["rbf", "sigmoid"]
]

grid_search = GridSearchCV(clf, param_grid, cv=3)
grid_search.fit(X, y)
```

8.4 커널 PCA



8.5 LLE

지역 선형 임베딩(LLE; Locally Linear Embedding)

이전과는 달리 “투영”에 의존하지 않는 매니폴드 학습

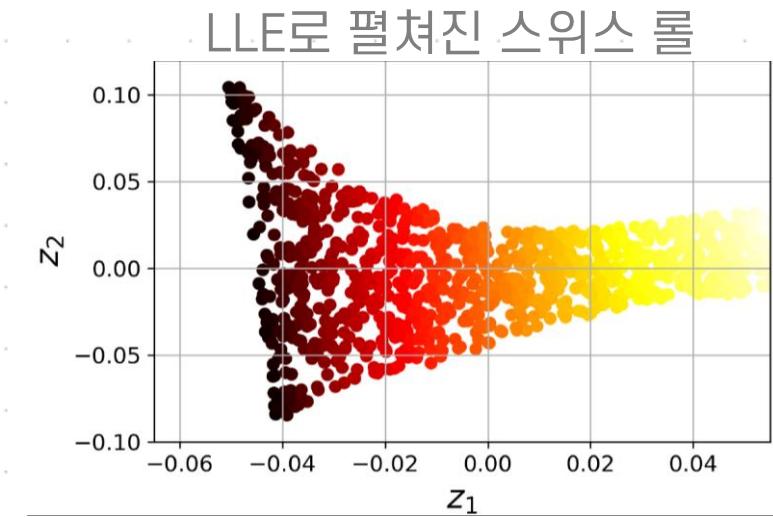
1) 각 훈련샘플이 가장 가까운 이웃(c.n.)에 얼마나 선형적으로 연관되어 있는지를 측정한다.

2) 국부적 관계가 가장 잘 보존되는 훈련 세트의 저차원 표현을 찾는다.

전체의 어느 한 부분에만 한정되는. 또는 그런 것(반대: 일반적)

```
from sklearn.manifold import LocallyLinearEmbedding
```

```
lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10)  
X_reduced = lle.fit_transform(X)
```



8.5 LLE

단계1: 선형적인 지역 관계 모델링

- 각 훈련 샘플 $x^{(i)}$ 에 대해 가장 가까운 k개의 샘플을 찾는다.
- 이 이웃샘플에 대한 선형 함수로 $x^{(i)}$ 를 재구성한다.

$$\widehat{\mathbf{W}} = \underset{\mathbf{W}}{\operatorname{argmin}} \sum_{i=1}^m \left(\mathbf{x}^{(i)} - \sum_{j=1}^m w_{i,j} \mathbf{x}^{(j)} \right)^2$$

조건
$$\begin{cases} w_{i,j} = 0 & \text{if } \mathbf{x}^{(j)} \text{ is not one of the } k \text{ c.n. of } \mathbf{x}^{(i)} \\ \sum_{j=1}^m w_{i,j} = 1 & \text{for } i = 1, 2, \dots, m \end{cases}$$

- 단계1을 거치면, 가중치 행렬 $\widehat{\mathbf{W}}$ 은 훈련샘플 사이에 있는 지역 선형관계를 띤

8.5 LLE

단계2: 관계를 보존하는 차원 축소

- 가능한 한 지역 선형관계가 보존되도록 훈련 샘플을 d차원 공간 ($d < n$)으로 매핑한다.
- 가중치를 고정하고 저차원의 공간에서 샘플 이미지의 최적 위치를 찾는다.
- 복잡도의 측면에서, 대량의 데이터셋에 이를 적용하기는 어렵다는 한계가 존재

$$\widehat{\mathbf{Z}} = \underset{\mathbf{Z}}{\operatorname{argmin}} \sum_{i=1}^m \left(\mathbf{z}^{(i)} - \sum_{j=1}^m \widehat{w}_{i,j} \mathbf{z}^{(j)} \right)^2$$

| 8.6 다른 차원 축소 기법

[1] 랜덤 투영(random projection)

랜덤한 선형 투영을 사용하여 데이터를 저차원 공간으로 투영

→ 실제로 거리를 잘 보존하는 것으로 밝혀짐

by William B. Johnson, and Joram Lindenstrauss

→ 품질은 샘플 수 / 목표 차원수에 따라 다름

- `sklearn.random_projection`

| 8.6 다른 차원 축소 기법

[2] **다차원 스케일링**(MDS; multi-dimensional scaling)

샘플 간의 거리를 보존하면서 차원을 축소

[3] **Isomap**

각 샘플을 가장 가까운 이웃과 연결하는 방식으로 그래프를 생성함

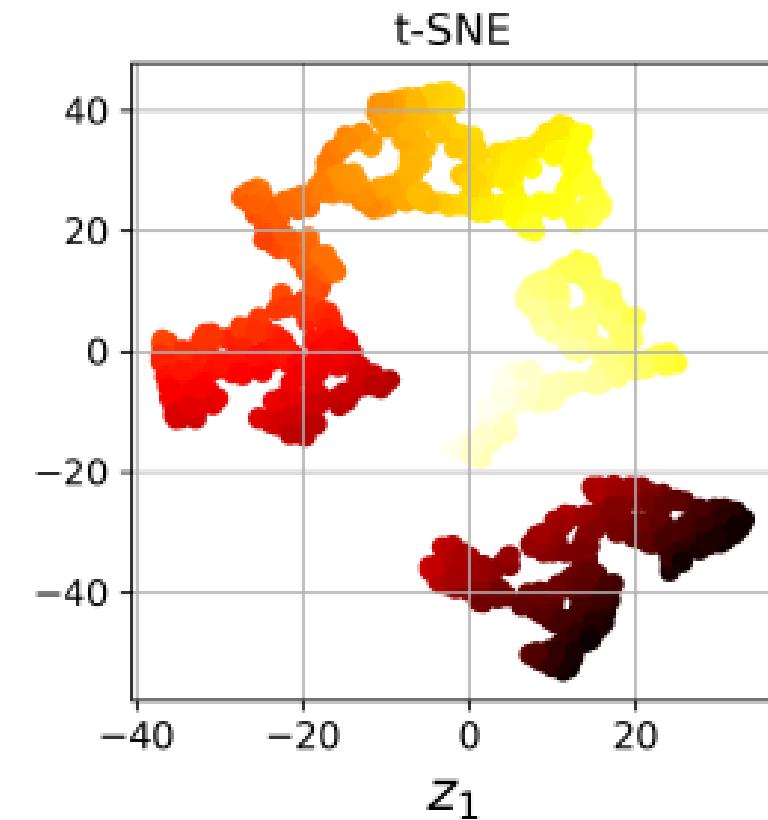
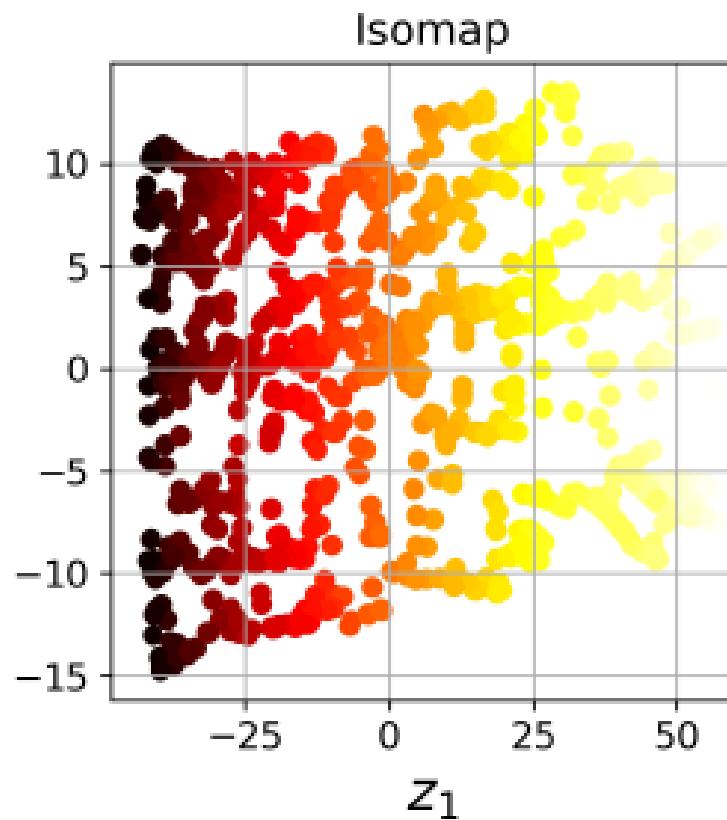
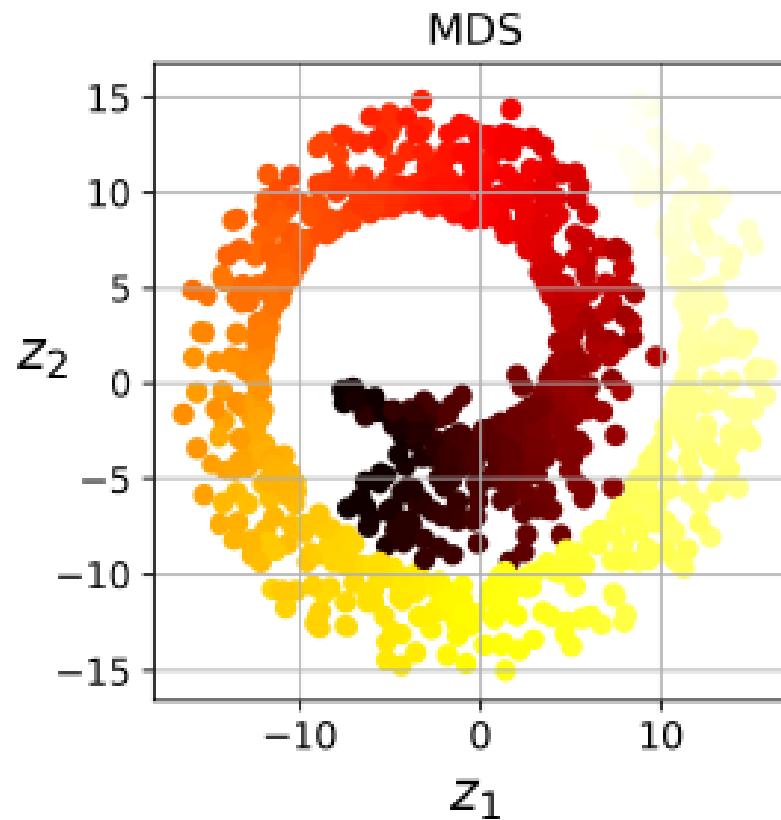
→ 샘플 간의 geodesic distance를 유지하며 차원을 축소해나감

[4] **t-SNE**(t-distributed stochastic neighbor embedding)

비슷한 샘플은 가까이, 비슷하지 않은 샘플은 멀리 떨어지도록 하며 차원축소

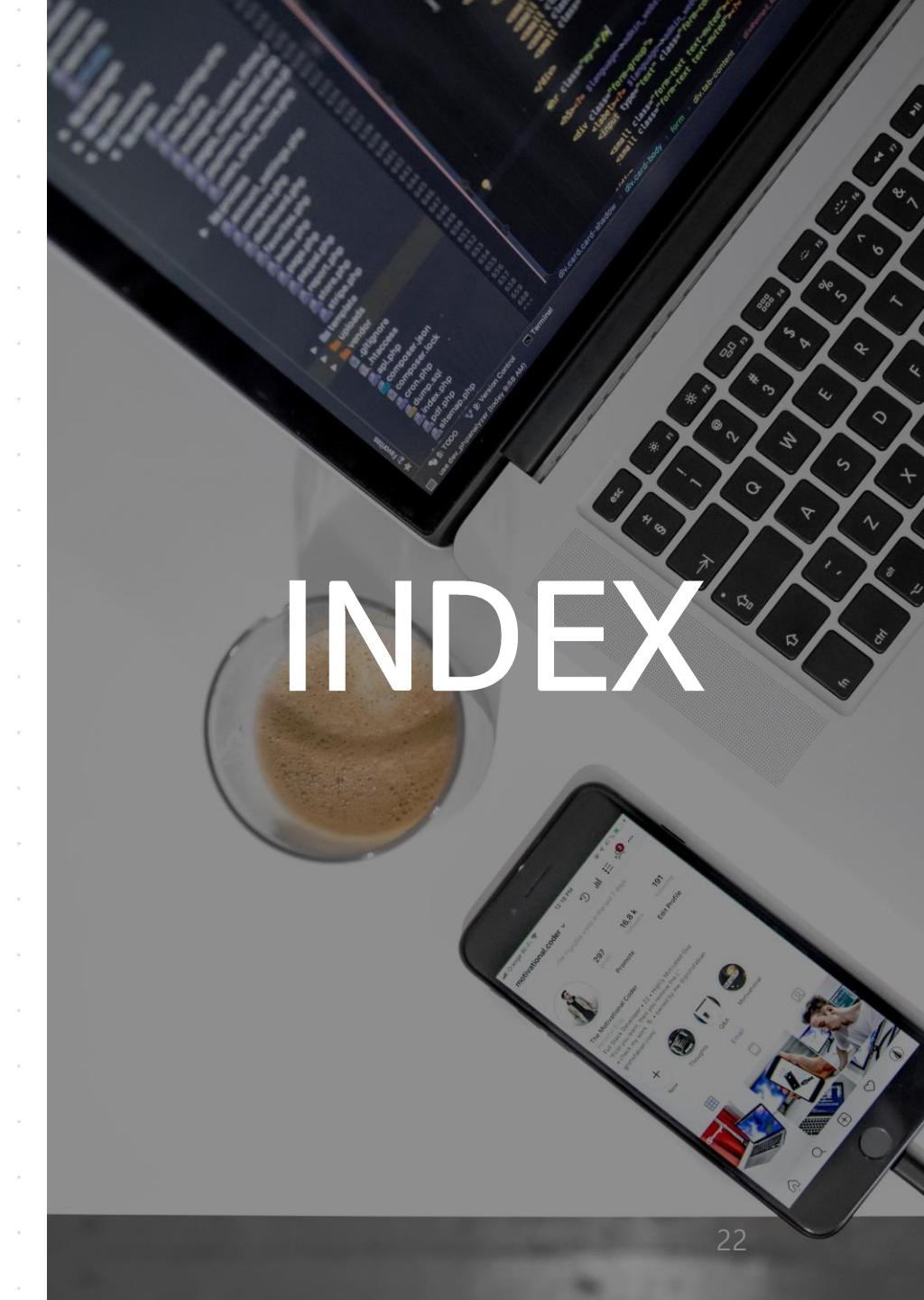
→ 시각화에 많이 사용, 고차원 공간의 샘플의 군집을 시각화할 때 사용

| 8.6 다른 차원 축소 기법



| 9.1 군집

| 9.2 가우시안 혼합



Intro

- 군집(clustering)

비슷한 샘플을 하나의 클러스터(cluster)로 묶어 모으는 것

e.g. 데이터 분석, 고객 분류, 추천 시스템, 검색 엔진 등

- 이상치 탐지(outlier detection)

“정상” 데이터가 어떻게 보이는지를 학습함

→ 이후 비정상 샘플을 감지하는데 사용한다.

e.g. 제조라인 결함 탐지, 시계열 데이터에서 새로운 경향성 찾기

- 밀도 추정(density estimation)

데이터셋 생성 확률 과정(random process)의 확률밀도함수를 추정하는 것

e.g. 이상치 탐지(→ 밀도가 매우 낮은 영역에 높은 샘플이 이상치)

9.1 군집

k-평균 (=Lloyd & Forgy algorithm)

반복 몇 번으로 레이블이 없는 데이터셋을 빠르고 효율적으로
클러스터 단위로 묶을 수 있는 알고리즘

```
from sklearn.cluster import KMeans
```

```
k = 5
```

```
kmeans = KMeans(n_clusters=k)
```

```
y_pred = kmeans.fit_predict(X)
```

알고리즘이 찾을 클러스터의 개수 **k**를 지정해야 한다.

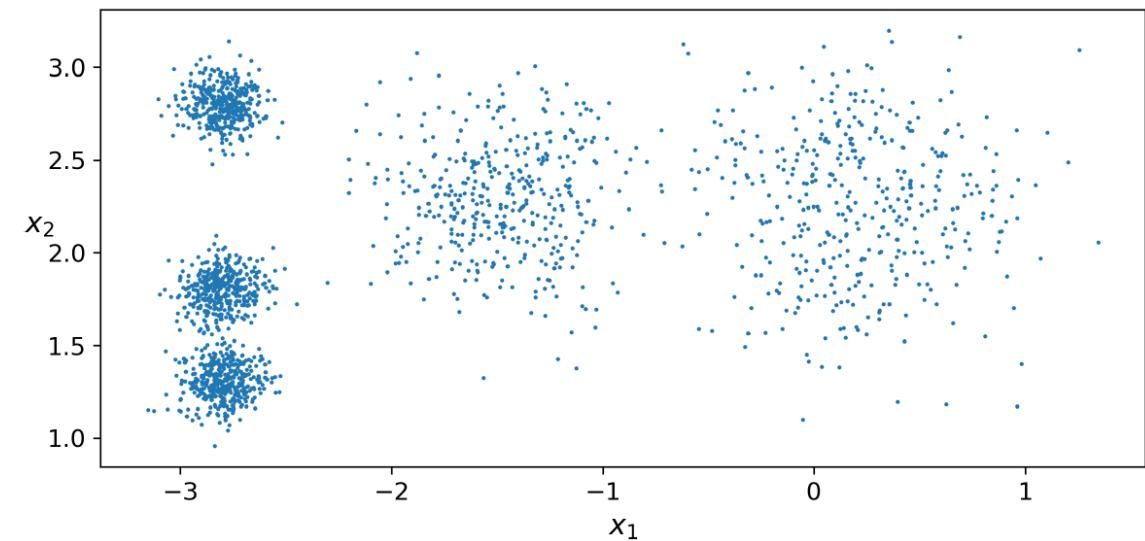
각 샘플은 k개의 클러스터 중 하나에 할당된다.

```
>>> y_pred
```

```
array([4, 0, 1, ..., 2, 1, 0], dtype=int32)
```

```
>>> y_pred is kmeans.labels_
```

```
True
```



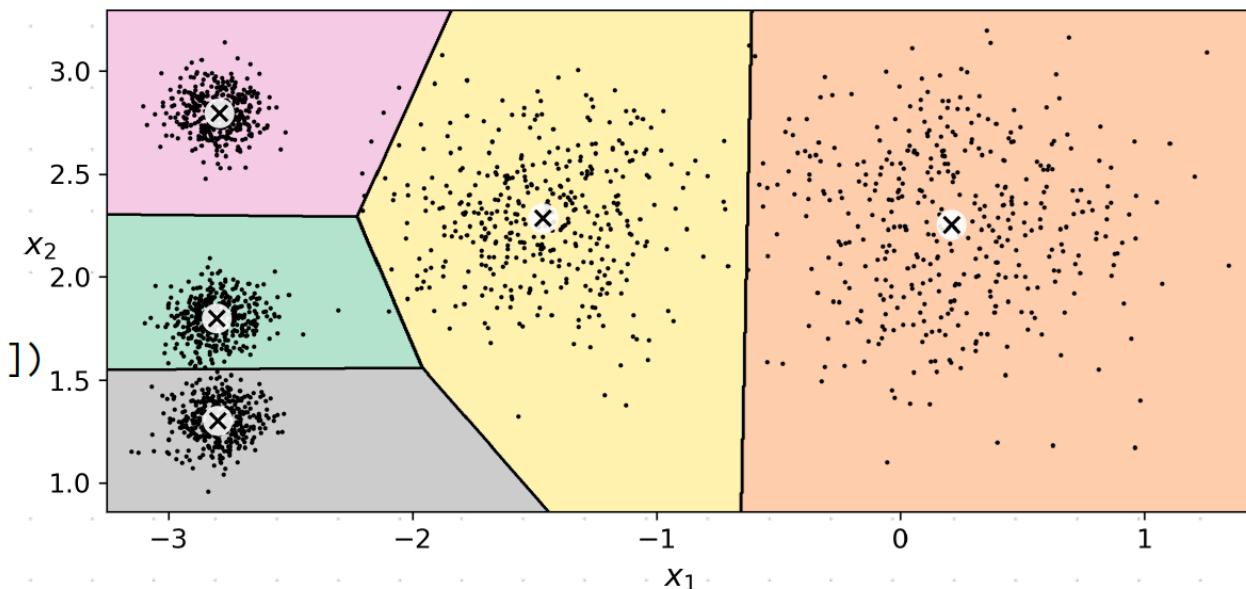
9.1 군집

k-평균(=Lloyd & Forgy algorithm)

- 센트로이드: 중심이 되는 특정 포인트
- 새로운 샘플에 가장 가까운 센트로이드의 클러스터를 할당할 수 있다.

```
>>> kmeans.cluster_centers_
array([[-2.80389616,  1.80117999],
       [ 0.20876306,  2.25551336],
       [-2.79290307,  2.79641063],
       [-1.46679593,  2.28585348],
       [-2.80037642,  1.30082566]])
```

```
>>> X_new = np.array([[0, 2], [3, 2], [-3, 3], [-3, 2.5]])
>>> kmeans.predict(X_new)
array([1, 1, 2, 2], dtype=int32)
```



클러스터의 크기가 많이 다르면 잘 작동하지 않음.

샘플을 클러스터에 할당할 때 센트로이드까지의 거리를 고려하는 것이 전부이기 때문
(그림에서 센트로이드는 x표로 표시함)

9.1 군집

k-평균 (=Lloyd & Forgy algorithm)

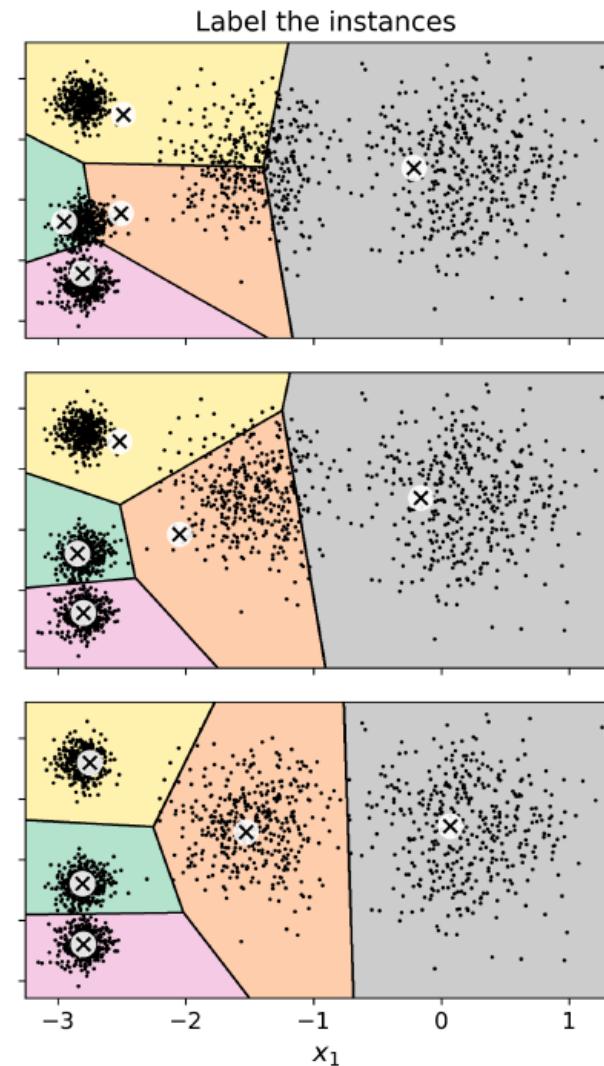
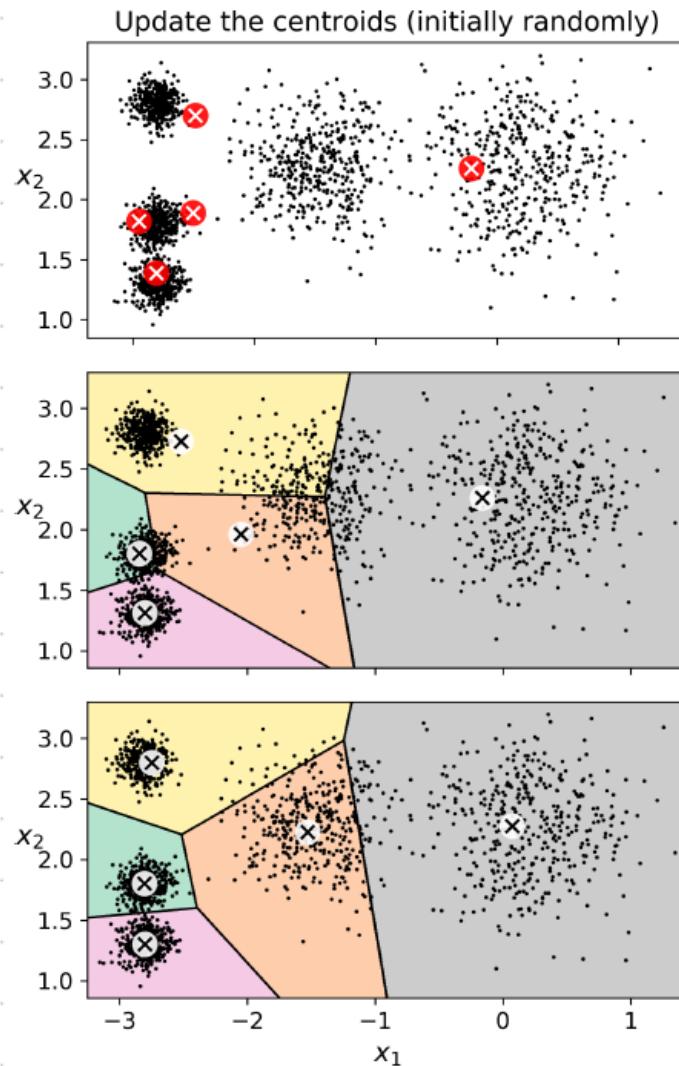
Q. 어떤 원리로 작동하는 것일까?

- 센트로이드가 주어진 경우: 모든 샘플에 가장 가까운 센트로이드를 선택
- 샘플의 label이 주어진 경우: 각 클러스터에 속한 샘플의 평균을 계산
 - 센트로이드를 쉽게 계산 가능함
- label이나 센트로이드가 주어지지 않은 경우
 - : 초기에는 센트로이드를 랜덤하게 선정
 - 샘플에 label을 할당하고 센트로이드를 업데이트하는데,
센트로이드에 변화가 더뎌질 때까지 계속 이를 반복함.

9.1 군집

k-평균 (=Lloyd & Forgy algorithm)

센트로이드 업데이트
(랜덤하게 초기화)



샘플에 label을 할당

9.1 군집

센트로이드 초기화 방법

- 1) 센트로이드 위치를 근사하게 알고 있다면 init 매개변수에 센트로이드 리스트를 담은 numpy 배열을 지정하고, n_init = 1로 설정

```
good_init = np.array([[-3, 3], [-3, 2], [-3, 1], [-1, 2], [0, 2]])  
kmeans = KMeans(n_clusters=5, init=good_init, n_init=1)
```

- 2) 랜덤 초기화를 다르게 하여 여러 번 알고리즘을 수행

- 최적의 솔루션을 측정하기 위한 성능지표: 이너셔(inertia)
→ 각 샘플과 가장 가까운 센트로이드 사이의 평균 제곱 거리

```
>>> kmeans.inertia_  
211.59853725816856
```

9.1 군집

k-평균 속도 개선과 미니배치 k-평균

삼각 부등식을 이용하여 불필요한 거리 계산을 피함으로써 속도를 개선함

→ KMeans 객체에서 기본적으로 사용함

→ MiniBatchKMeans 객체에서는 각 반복마다 전체 데이터셋 대신

미니배치를 사용하여 센트로이드를 조금씩 이동시키는 방식으로 3~4배 속도개선

→ 응용: 메모리에 들어가지 않는 대량의 데이터셋에 적용이 가능하다.

```
from sklearn.cluster import MiniBatchKMeans
```

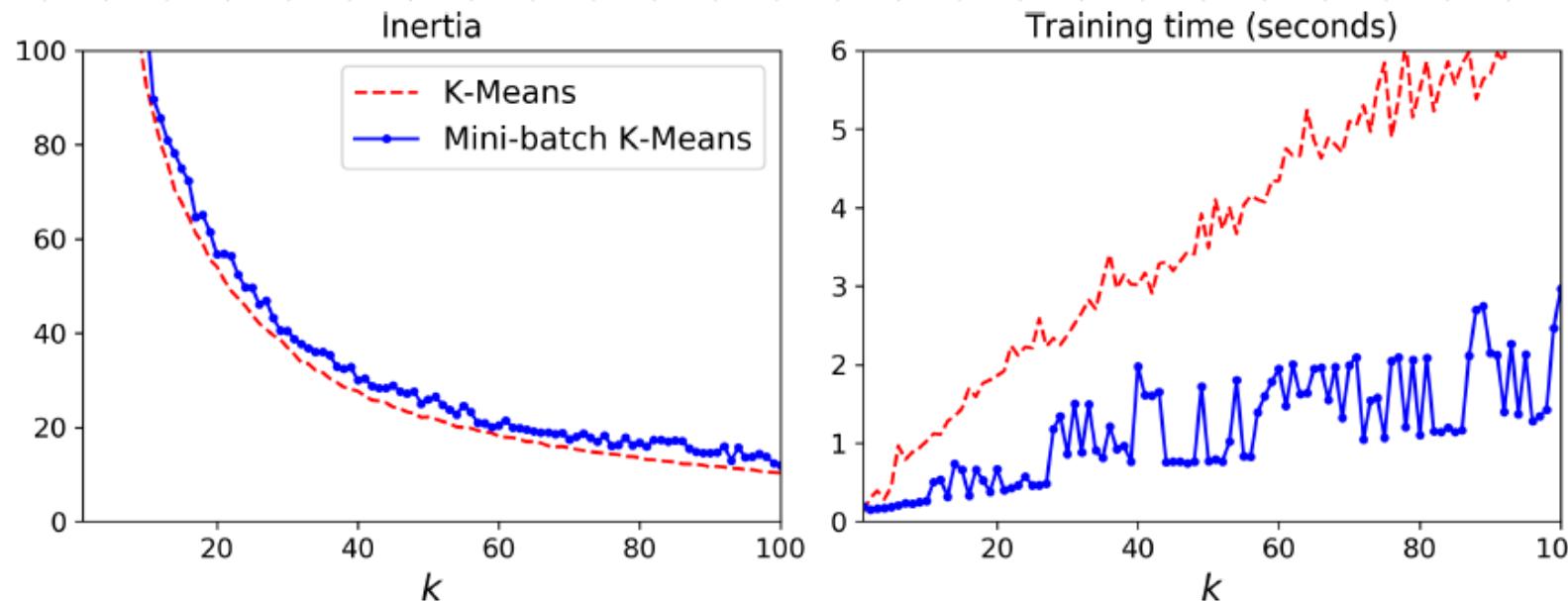
```
minibatch_kmeans = MiniBatchKMeans(n_clusters=5)
minibatch_kmeans.fit(X)
```

9.1 군집

k-평균 속도 개선과 미니배치 k-평균

Q. 그렇다면 성능은 어떨까?

- 미니배치 k-평균 알고리즘이 일반 k-평균 알고리즘보다 훨씬 빠르다.
- 이너셔는 일반적으로 미니배치 k-평균 알고리즘이 조금 더 나쁘다.
(특히 클러스터 개수 k 가 증가할 때 더욱 심화됨)



9.1 군집

최적의 클러스터 개수 찾기

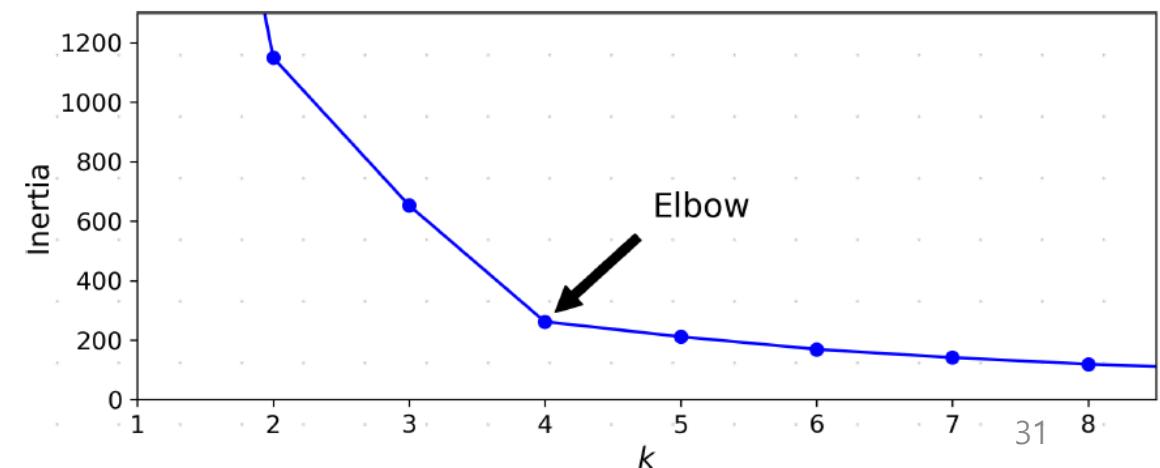
1) inertia의 추세로 어림잡기

X: $k \rightarrow$ Y: inertia의 함수를 그려보았을 때

k 가 4까지 증가할 때 빠르게 줄어들다가 그 이후로는 느리게 감소한다.

→ 그래프 상으로 이 지점을 Elbow라고 일컫는다.

(다소 엉성한 방법이 될 수도 있음)



9.1 군집

최적의 클러스터 개수 찾기

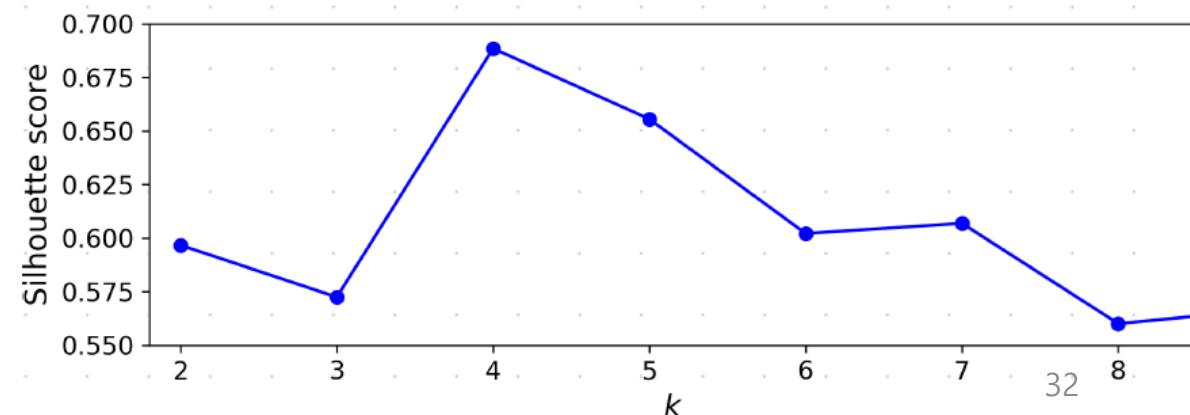
2) 실루엣 점수(silhouette score)

모든 샘플에 대한 실루엣 계수(silhouette coefficient)의 평균

$$(\text{실루엣 계수}) = (b - a) / \max(a, b) \quad (\text{범위: } -1 \leq \text{실루엣 계수} \leq +1)$$

- * 잘못된 클러스터에 할당됨 → 클러스터의 경계 → 클러스터 안쪽에 잘 위치
- * a: 동일한 클러스터에 있는 다른 샘플까지 평균 거리(클러스터 내부 평균거리)
- * b: 가장 가까운 클러스터까지 평균 거리

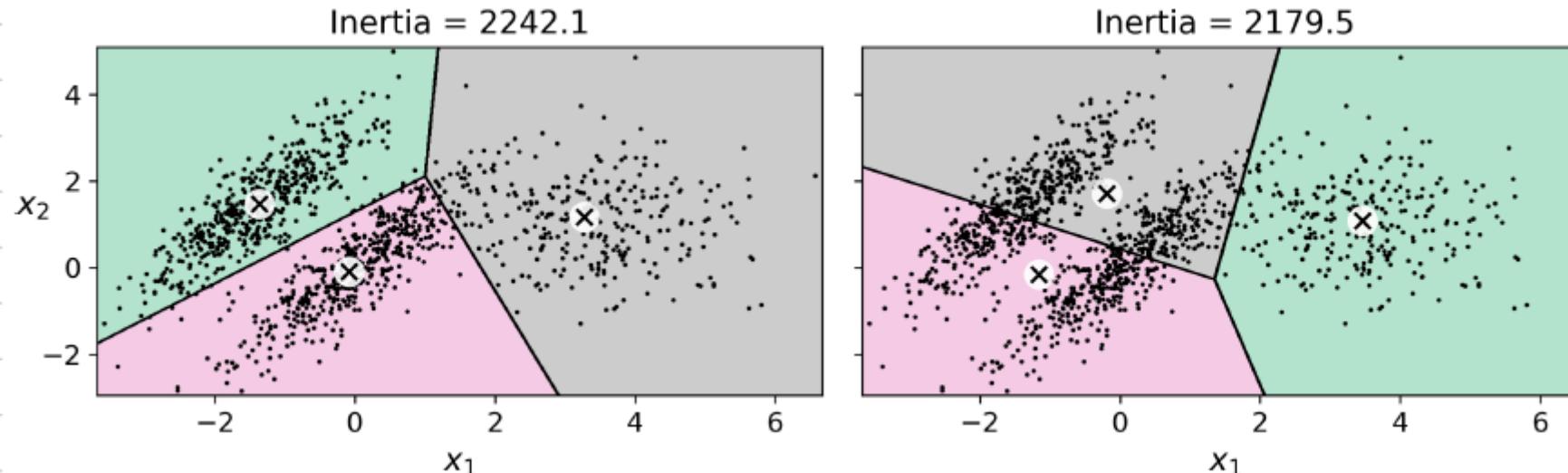
```
>>> from sklearn.metrics import silhouette_score  
>>> silhouette_score(X, kmeans.labels_)  
0.655517642572828
```



9.1 군집

k-평균의 한계

- 최적의 솔루션에 이르기 위해서는 알고리즘이 여러 번 반복되어야 한다.
- 클러스터의 개수를 지정해야 한다.
- Cluster의 크기나 밀집도가 서로 다르거나 원형(spherical)이 아니면 잘 동작하지 않을 수 있다.(아래의 그림은 타원형 cluster)



9.1 군집

군집을 사용한 이미지 분할

- 이미지 분할(image segmentation)

이미지를 여러 개의 세그먼트로 분할하는 작업

- 시맨틱 분할(semantic segmentation)

동일한 종류의 물체에 속한 모든 픽셀은 같은 세그먼트에 할당됨

e.g. imread() 함수를 이용하여 이미지 읽기

```
>>> from matplotlib.image import imread # you could also use `imageio.imread()`
>>> image = imread(os.path.join("images", "clustering", "ladybug.png"))
>>> image.shape
(533, 800, 3)      이미지는 3D 배열로 표현됨
                           각각 높이, 너비, 컬러 채널 개수(RGB 채널)
```

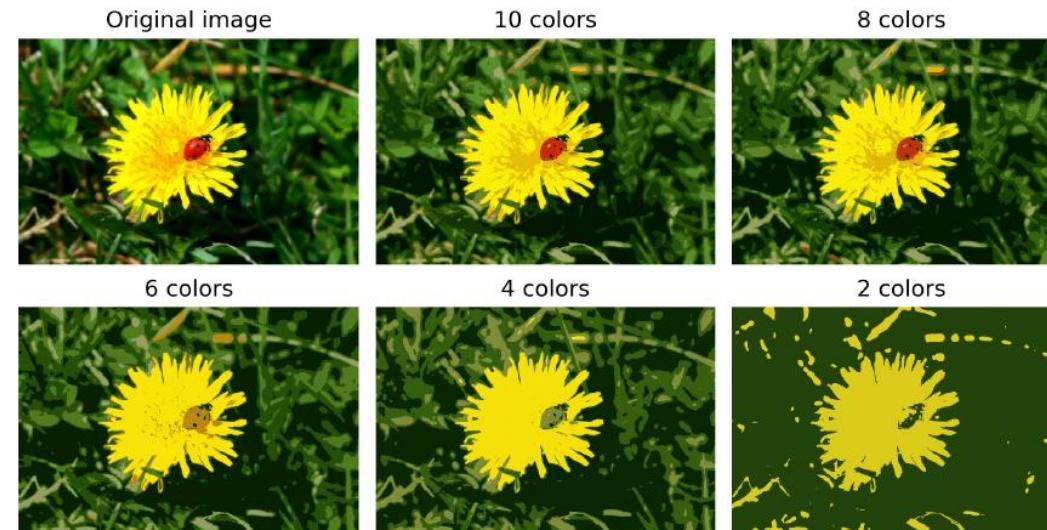
9.1 군집

군집을 사용한 이미지 분할

RGB 색상의 긴 리스트로 변환 후 k-평균을 사용해

이 색상을 클러스터로 모으는 코드

```
X = image.reshape(-1, 3)
kmeans = KMeans(n_clusters=8).fit(X)
segmented_img = kmeans.cluster_centers_[kmeans.labels_]
segmented_img = segmented_img.reshape(image.shape)
```



9.1 군집

군집을 사용한 전처리

파이프라인을 만들어 훈련세트를 50개의 cluster로 모은 후

이미지를 50개 cluster까지의 거리로 바꾼 후 로지스틱 회귀 모델 적용

```
from sklearn.pipeline import Pipeline

pipeline = Pipeline([
    ("kmeans", KMeans(n_clusters=50)),
    ("log_reg", LogisticRegression()),
])
pipeline.fit(X_train, y_train)

>>> pipeline.score(X_test, y_test)
0.9822222222222222
```

train_test_split으로 데이터셋 나누고 로지스틱 회귀만 적용했을 때는 96.9% 정확도를 가짐

데이터셋의 차원을 감소시켰으나
성능 향상은 대부분 변환된 데이터셋이 원본 데이터셋보다
선형적으로 더 잘 구분할 수 있기 때문이다.

9.1 군집

군집을 사용한 준지도 학습

label이 없는 데이터가 많고 label이 있는 데이터는 적을 때 사용한다.

MNIST에서 훈련 세트를 50개의 cluster로 모으고, 각 클러스터에서
센트로이드에 가장 가까운 이미지를 찾는다. (→ 대표 이미지; representative image)

```
k = 50
kmeans = KMeans(n_clusters=k)
X_digits_dist = kmeans.fit_transform(X_train)
representative_digit_idx = np.argmin(X_digits_dist, axis=0)
X_representative_digits = X_train[representative_digit_idx]
```

4	8	0	6	8	3	7	7	9	1
5	5	8	5	1	1	2	5	6	1
1	6	9	0	8	3	0	7	4	1
6	5	2	4	1	8	6	3	9	2
4	2	9	4	7	6	2	3	1	1

9.1 군집

DBSCAN

밀집된 연속적 지역을 클러스터로 정의한다.

- i) 각 샘플에서 작은 거리 ϵ 이내에 있는 샘플의 개수를 센다($\rightarrow \epsilon\text{-이웃}$)
- ii) $\epsilon\text{-이웃}$ 내에 적어도 min_samples 개 샘플이 있다면 이를 핵심 샘플로 간주
 \rightarrow 핵심 샘플: 밀집된 지역에 있는 샘플
- iii) 핵심 샘플의 이웃에 있는 모든 샘플은 동일한 클러스터에 속함
 \rightarrow 이 규칙은 연쇄적으로 다른 이웃들에게도 계속해서 적용되어 클러스터를 형성
- iv) 핵심 샘플이 아니며 & 이웃이 아닌 샘플은 “이상치”로 판단

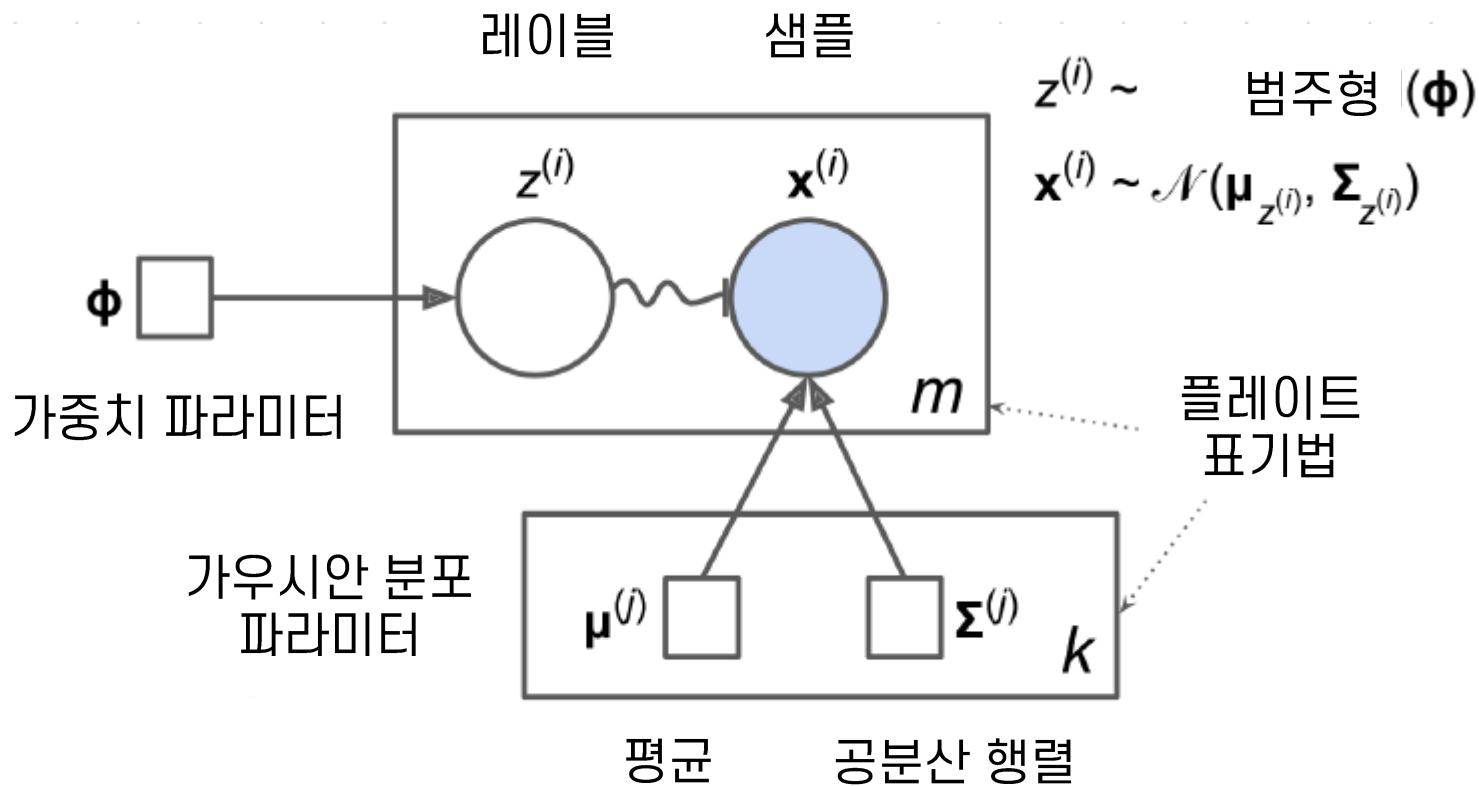
| 9.2 가우시안 혼합

가우시안 혼합 모델(GMM; Gaussian mixture model)

샘플이 파라미터가 알려지지 않은 여러 개의 혼합된 가우시안 분포에서 생성되었다고 가정하는 확률모델

- 1개의 가우시안 분포에서 생성된 모든 샘플은 하나의 Cluster를 형성
- Cluster는 타원형, 각각은 모양/크기/밀집도/방향이 다름
- GaussianMixture 객체 → 가우시안 분포 개수 k가 필요함

9.2 가우시안 혼합



- 원은 확률변수를 나타낸다.
- 사각형은 고정값을 나타낸다.
(= 모델의 파라미터)
- 큰 사각형 → 플레이트(plate)
→ 이 안의 내용이 여러 번 반복됨
- 플레이트 우하단의 숫자
= 플레이트 안의 내용의 반복 횟수

9.2 가우시안 혼합

데이터셋 X 가 주어지면 가중치 Φ 와 전체 분포의 파라미터 $\mu^{(1)}$ 에서 $\mu^{(k)}$ 까지와 $\Sigma^{(1)}$ 에서 $\Sigma^{(k)}$ 까지를 추정한다. → GaussianMixture 객체

```
from sklearn.mixture import GaussianMixture
gm = GaussianMixture(n_components=3, n_init=10)
gm.fit(X)

>>> gm.weights_
array([0.20965228, 0.4000662 , 0.39028152])
>>> gm.means_
array([[ 3.39909717,  1.05933727],
       [-1.40763984,  1.42710194],
       [ 0.05135313,  0.07524095]])
>>> gm.covariances_
array([[[ 1.14807234, -0.03270354],
       [-0.03270354,  0.95496237]],
       [[ 0.63478101,  0.72969804],
       [ 0.72969804,  1.1609872 ]],
       [[ 0.68809572,  0.79608475],
       [ 0.79608475,  1.21234145]]])
```

9.2 가우시안 혼합

하드 군집: 새로운 샘플을 가장 비슷한 클러스터에 할당 → predict()

소프트 군집: 새로운 샘플이 특정 클러스터에 속할 확률을 예측 → predict_proba()

```
>>> gm.predict(X)
array([2, 2, 1, ..., 0, 0, 0])
>>> gm.predict_proba(X)
array([[2.32389467e-02, 6.77397850e-07, 9.76760376e-01],
       [1.64685609e-02, 6.75361303e-04, 9.82856078e-01], ...,
       [2.01535333e-06, 9.99923053e-01, 7.49319577e-05],
       [9.99999571e-01, 2.13946075e-26, 4.28788333e-07],
       [1.00000000e+00, 1.46454409e-41, 5.12459171e-16],
       [1.00000000e+00, 8.02006365e-41, 2.27626238e-15]])
```

생성 모델(generative model) → 이 모델에서 새로운 샘플을 만들 수 있다.

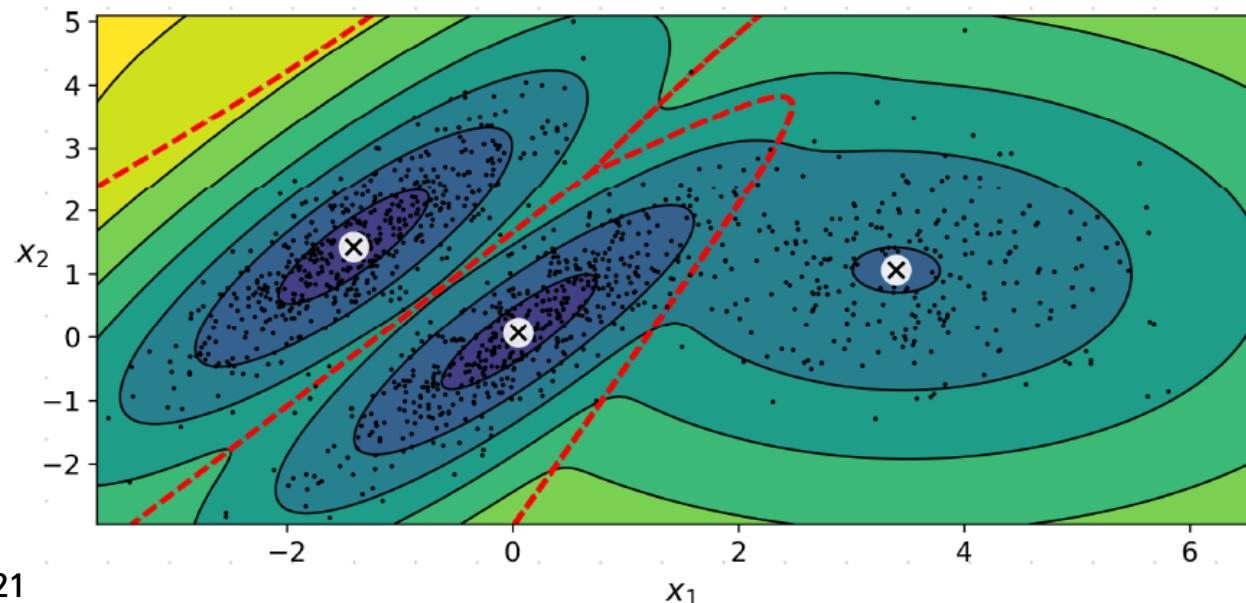
```
>>> X_new, y_new = gm.sample(6)
>>> X_new
array([[ 2.95400315,  2.63680992],
       [-1.16654575,  1.62792705],
       [-1.39477712, -1.48511338],
       [ 0.27221525,  0.690366 ],
       [ 0.54095936,  0.48591934],
       [ 0.38064009, -0.56240465]])
>>> y_new
array([0, 1, 2, 2, 2, 2])
```

9.2 가우시안 혼합

주어진 위치에서 모델의 밀도를 추정할 수도 있다.

→ `score_samples()` : 샘플이 주어지면 그 위치의 PDF의 로그값을 예측
(* PDF: 확률밀도함수; probability density function)

```
>>> gm.score_samples(X)  
array([-2.60782346, -3.57106041, -3.33003479, ..., -3.51352783,  
      -4.39802535, -3.80743859])
```



훈련된 가우시안 혼합 모델의
클러스터 평균, 결정 경계, 밀도 등고선을
나타낸 그림

| 9.2 가우시안 혼합

가우시안 혼합을 사용한 이상치 탐지

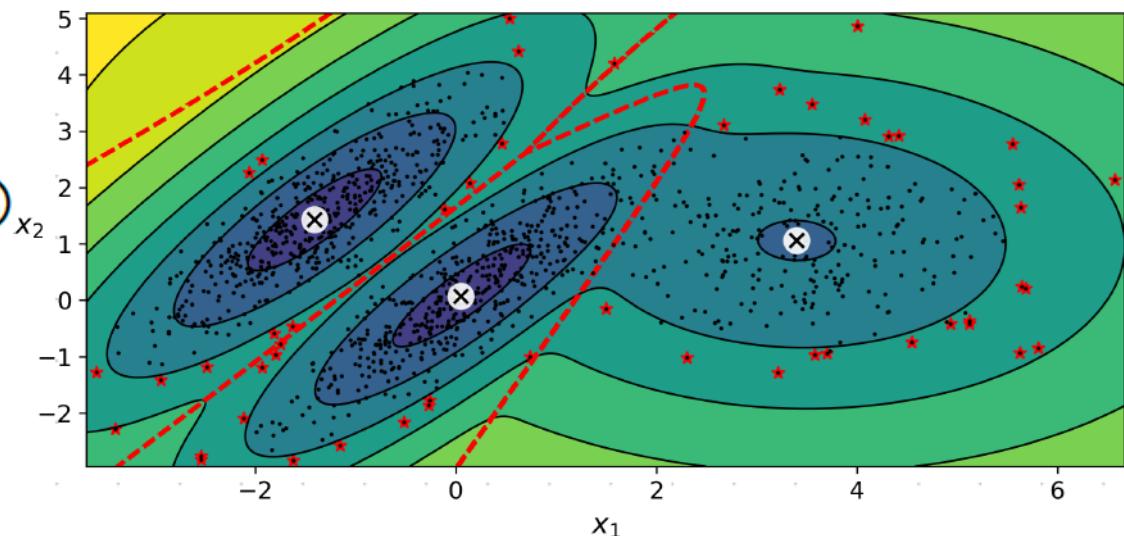
이상치 탐지: 보통과 많이 다른 샘플을 감지하는 작업 (반의어: 정상치)

in 가우시안 혼합 모델?

→ 밀도가 낮은 지역에 있는 모든 샘플을 이상치로 간주한다.

→ 밀도 임곗값(density threshold)을 지정해야 한다.

```
densities = gm.score_samples(X)
density_threshold = np.percentile(densities, 4)
anomalies = X[densities < density_threshold]
```



9.2 가우시안 혼합

클러스터 개수 선택하기

k-평균에서는 cluster 개수 선택을 위해 inertia나 실루엣 점수 등을 사용함

가우시안 혼합에서는 이러한 지표를 사용할 수 없음!

→ cluster가 타원형이거나 크기가 다를 때는 안정하지 못하기 때문임

- BIC; Bayesian Information Criterion

$$BIC = \log(m) p - 2 \log(\hat{L})$$

m: 샘플의 개수

p: 모델이 학습할 파라미터 개수

\hat{L} : 모델의 가능도 함수의 최댓값

- AIC; Akaike Information Criterion

$$AIC = 2p - 2 \log(\hat{L})$$

`>>> gm.bic(X)`

8189.74345832983

`>>> gm.aic(X)`

8102.518178214792

| 9.2 가우시안 혼합

베이즈 가우시안 혼합 모델

BayesianGaussianMixture 객체

최적의 cluster 개수를 수동으로 찾지 않고 불필요한 cluster의 가중치를 0으로 만든다.

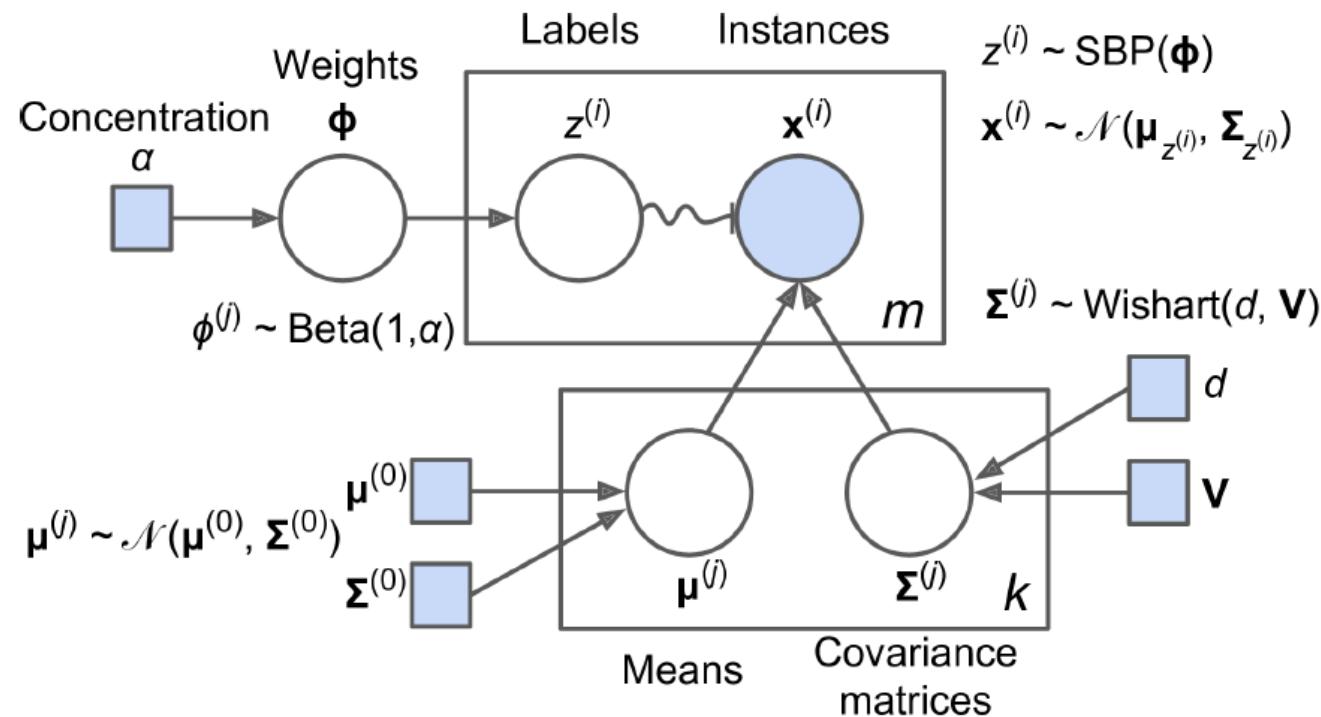
자동으로 불필요한 cluster를 삭제한다.

```
>>> from sklearn.mixture import BayesianGaussianMixture  
>>> bgm = BayesianGaussianMixture(n_components=10, n_init=10, random_state=42)  
>>> bgm.fit(X)  
>>> np.round(bgm.weights_, 2)  
array([0.4 , 0.21, 0.4 , 0. , 0. , 0. , 0. , 0. , 0. , 0. ])
```

알고리즘이 자동적으로 cluster가 3개가 필요한 상황이라는 것을 감지한 것임.

9.2 가우시안 혼합

베이즈 가우시안 혼합 모델



- a 가 클수록 ϕ 값이 0에 가까워지고 많은 클러스터를 만들게 된다.
- 파라미터 d, V 가 cluster의 분포 모양을 제어한다.

9.2 가우시안 혼합

베이즈 가우시안 혼합 모델

베이즈 정리: 데이터 X 를 관측한 후의 잠재 변수에 대한 확률 분포 업데이트
→ X 가 주어졌을 때 z 의 조건부 확률인 사후 확률분포 $P(z | X)$ 계산

$$p(z|X) = \text{(사후 확률)} = \frac{\text{(가능도)} \times \text{(사전 확률)}}{\text{(증거)}} = \frac{p(X|z)p(z)}{p(X)}$$

$$p(X) = \int p(X|z)p(z)dz$$

첫번째 식에서 분모에 해당하는 $p(X)$ 는 계산하기가 어려움
→ 변분 추론(variational inference)을 통해 해결

9.2 가우시안 혼합

베이즈 가우시안 혼합 모델

변분 추론: 자체적인 변분 파라미터 λ 를 가진 분포패밀리 $q(z; \lambda)$ 선택

→ $q(z)$ 가 $p(z | X)$ 의 좋은 근사값이 되도록 λ 를 최적화함

→ $q(z)$ 에서 $p(z | X)$ 로의 KL발산을 최소화하는 λ 값을 찾아 이를 해결

$$\begin{aligned} D_{KL}(q || p) &= \mathbb{E}_q \left[\log \frac{q(\mathbf{z})}{p(\mathbf{z} | \mathbf{X})} \right] \\ &= \mathbb{E}_q [\log q(\mathbf{z}) - \log p(\mathbf{z} | \mathbf{X})] \\ &= \mathbb{E}_q \left[\log q(\mathbf{z}) - \log \frac{p(\mathbf{z}, \mathbf{X})}{p(\mathbf{X})} \right] \\ &= \mathbb{E}_q [\log q(\mathbf{z}) - \log p(\mathbf{z}, \mathbf{X}) + \log p(\mathbf{X})] \\ &= \mathbb{E}_q [\log q(\mathbf{z})] - \mathbb{E}_q [\log p(\mathbf{z}, \mathbf{X})] + \mathbb{E}_q [\log p(\mathbf{X})] \\ &= \mathbb{E}_q [\log p(\mathbf{X})] - (\mathbb{E}_q [\log p(\mathbf{z}, \mathbf{X})] - \mathbb{E}_q [\log q(\mathbf{z})]) \\ &= \log p(\mathbf{X}) - \text{ELBO} \end{aligned}$$

where $\text{ELBO} = \mathbb{E}_q [\log p(\mathbf{z}, \mathbf{X})] - \mathbb{E}_q [\log q(\mathbf{z})]$

$\log[p(\mathbf{X})]$ 에서 증거 하한(ELBO)을 뺀 식으로
다시 쓸 수 있다.

이때 이것의 로그값은 q 에 의존하지 않고
상수항이므로 KL발산을 최소화하려면
ELBO를 최대화해야 한다.



A close-up photograph of a Christmas tree branch. The branch is adorned with numerous small, glowing ornaments in shades of blue, green, yellow, and red. The background is dark, making the bright lights stand out. The overall atmosphere is festive and celebratory.

Thank You