

# 핸즈온 머신러닝

## 8장. 차원 축소

박해선(웁긴이)

[haesun.park@tensorflow.blog](mailto:haesun.park@tensorflow.blog)

<https://tensorflow.blog>

# 케라스 창시자에게 배우는 딥러닝



창시자의 철학까지 담았다!

DEEP LEARNING  
WITH PYTHON



프랑스와 스페인  
박해선 옮김



딥러닝, 신경망, 머신 러닝 기초부터  
컴퓨터 비전과 텍스트, 시퀀스, 생성 모델을 위한 딥러닝까지  
입문하면서 알아야 할 모든 것을 설명한다!

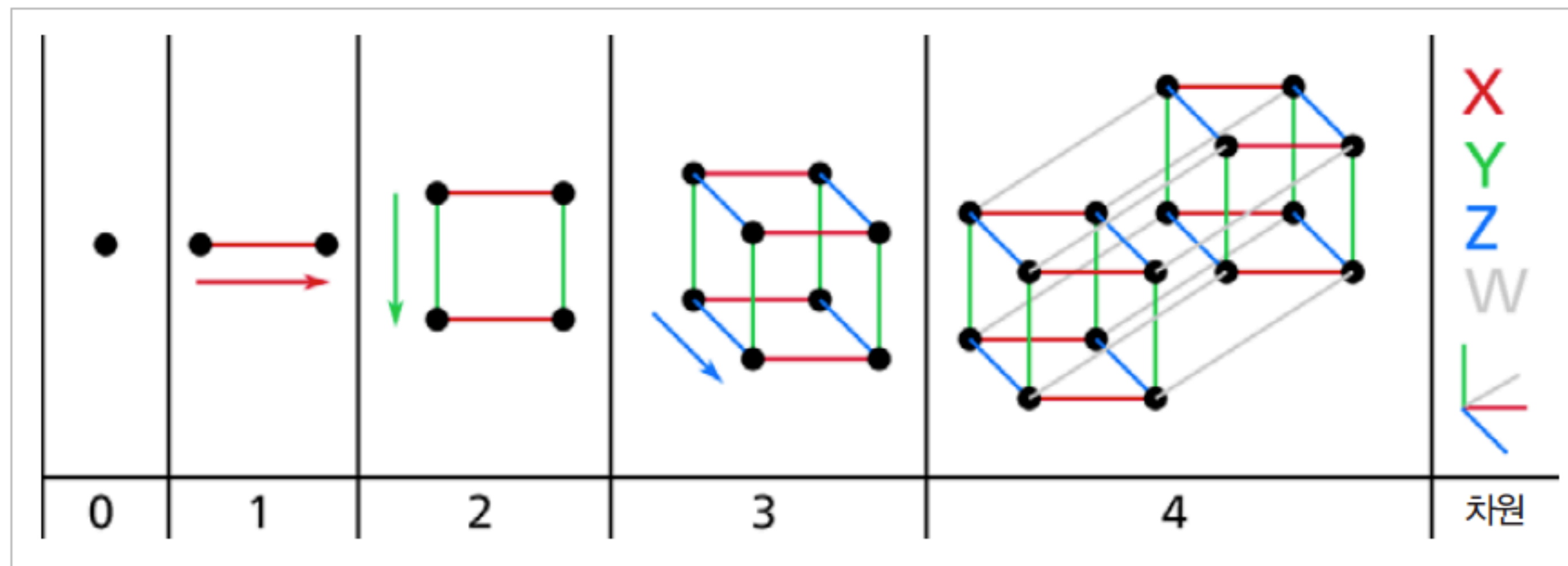
11월

# 차원의 축소

- 수천, 수백만 개의 특성이 훈련 속도를 낮추고 솔루션을 찾기 어렵게 만듭니다.
- 특성의 수를 줄이는 차원 축소(비지도 학습)로 해결 가능한 문제로 변경 가능합니다.
- 예를 들어, 이미지 주변 픽셀을 제거하거나 인접한 픽셀을 평균내어 합칩니다.
- 차원 축소는 일반적으로 훈련 속도는 높아지지만 성능은 낮아지고 작업 파이프라인이 복잡해 집니다. 드물게 성능이 높아지는 경우가 있습니다(이상치 제거).
- 차원 축소는 데이터 시각화에도 유용합니다.
- PCA, 커널 PCA, LLE를 다룹니다.

# 차원의 저주

- 사람은 4차원 초입방체도 상상하기 어렵습니다.

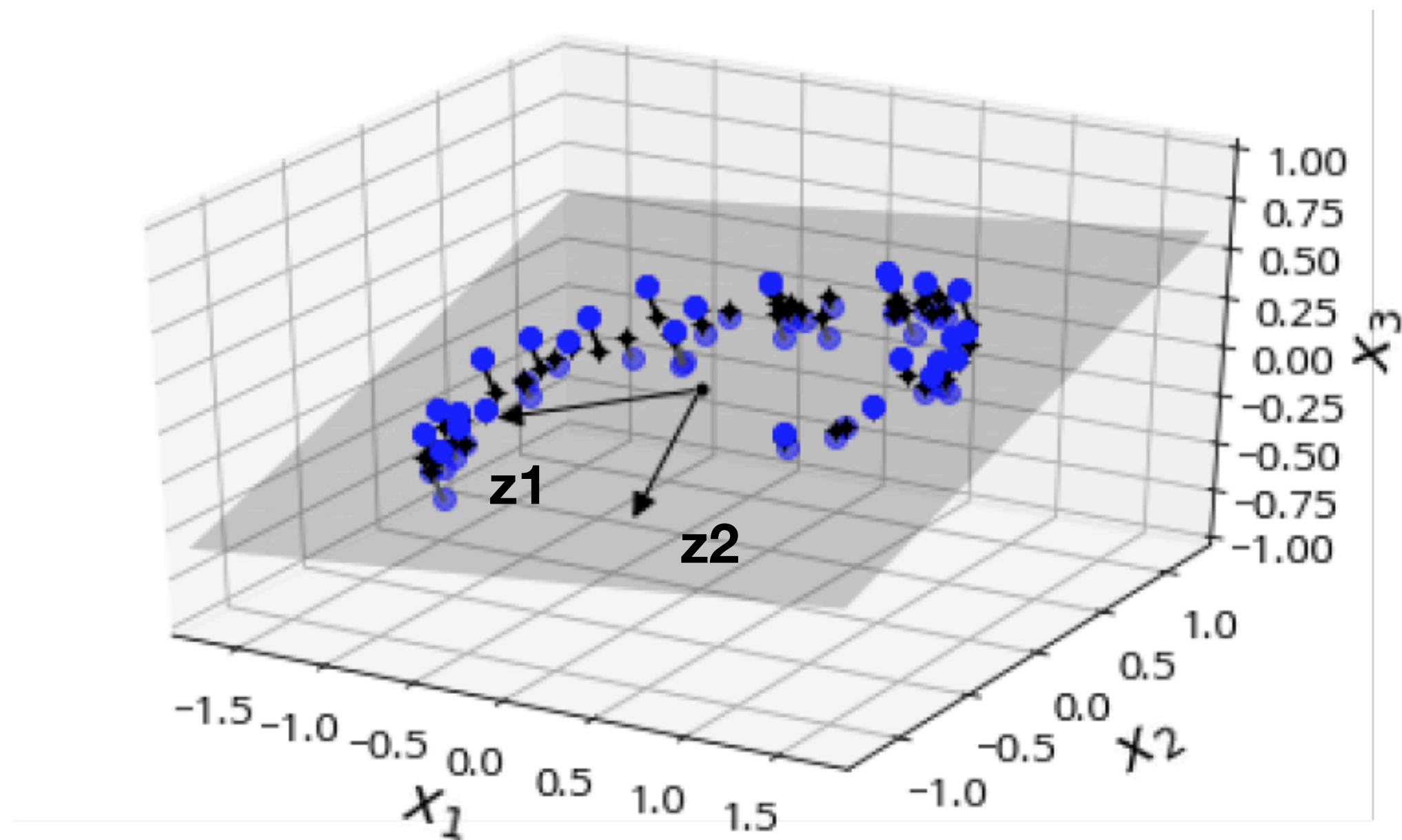


- 1x1 사각형에서 0.001 이내의 면적은  $1-(1-0.001 \times 2)^2=0.003996$ 이므로 약 4%(사이 거리: 0.52)
- 10,000 차원 초입방체는  $1-(1-0.001 \times 2)^{10000}=0.9999999998$ 이므로 99.99%(사이 거리: 428.25)
- 고차원의 대부분의 점들은 경계에 가까이 위치해 있습니다.

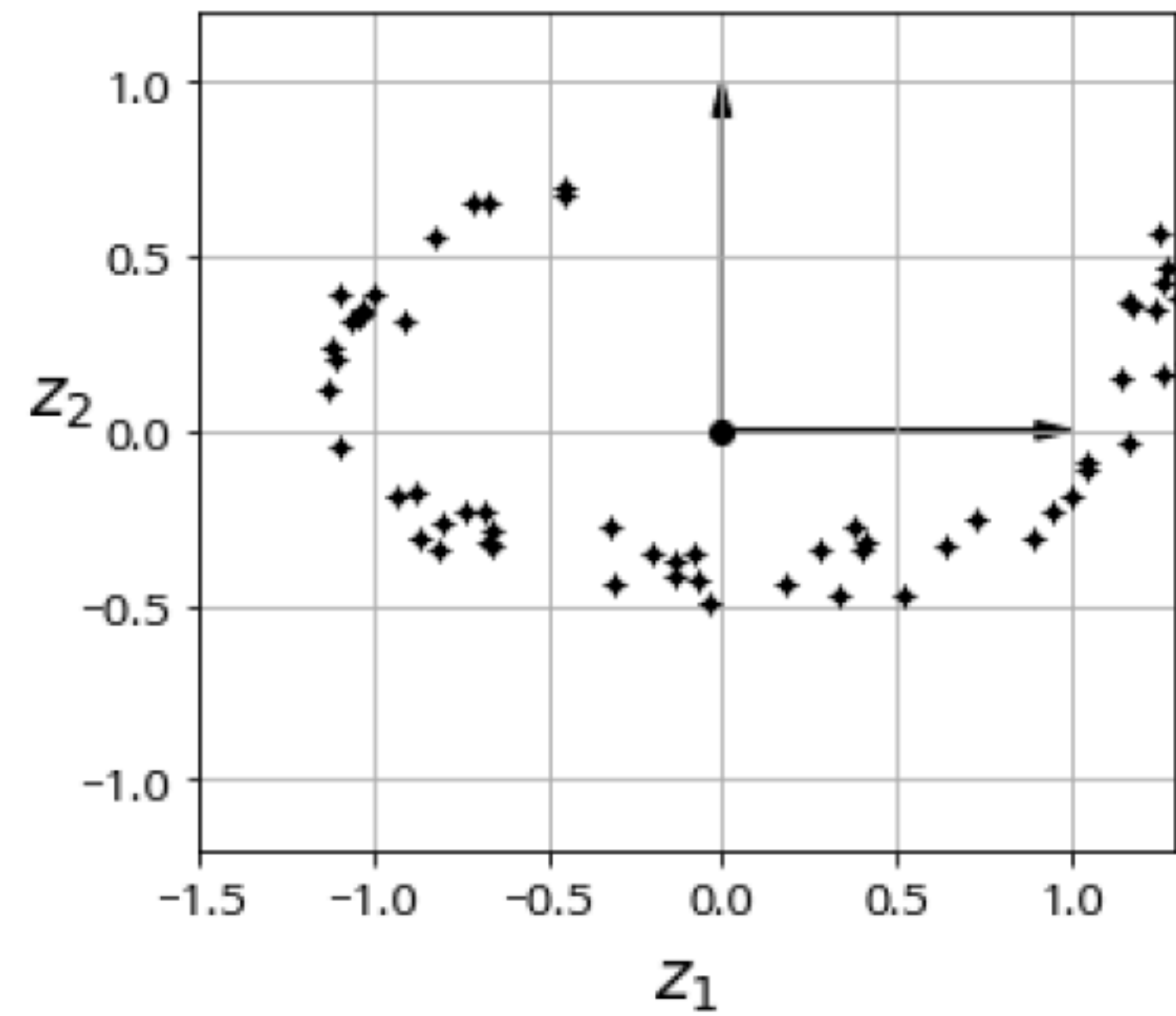
# 데이터 밀도

- 고차원 데이터셋은 샘플 간의 거리가 멍니다. 즉 희박합니다.
- 새로운 샘플에 대해 예측하려면 많은 보간이 필요합니다(과대적합 위험).
- 간단한 해결 방법은 데이터 밀도가 충분해질 때까지 데이터를 더 모으는 것입니다.
- 100개의 특성이 있는 경우 샘플 간의 거리를 0.1 이내로 하려면  $10^{100}$ 개가 필요합니다. 이는 우주 전체에 있는 원자수( $10^{80}$ )보다 많습니다.

# 투영

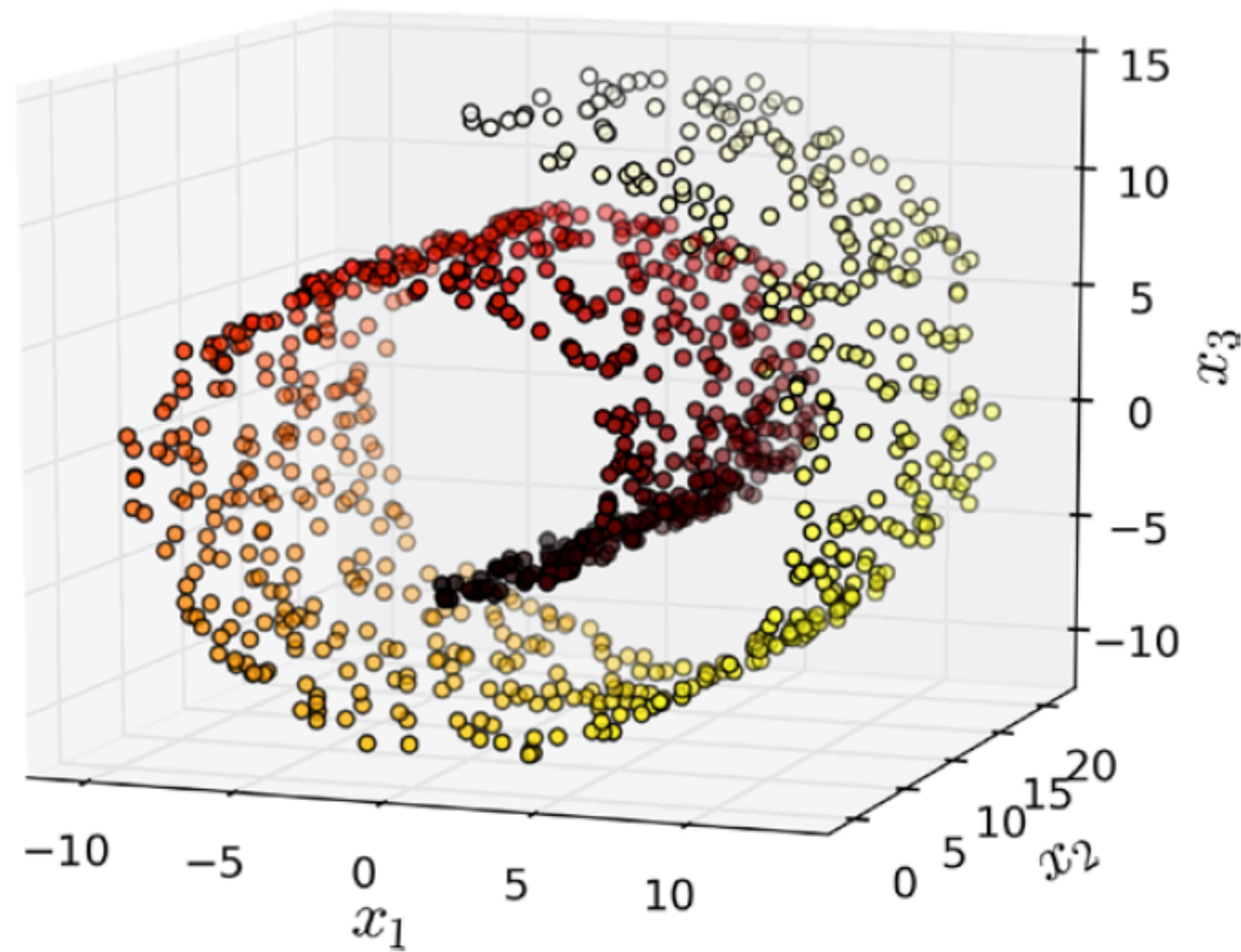


고차원 공간을  
저차원 부분 공간으로  
수직으로 투영

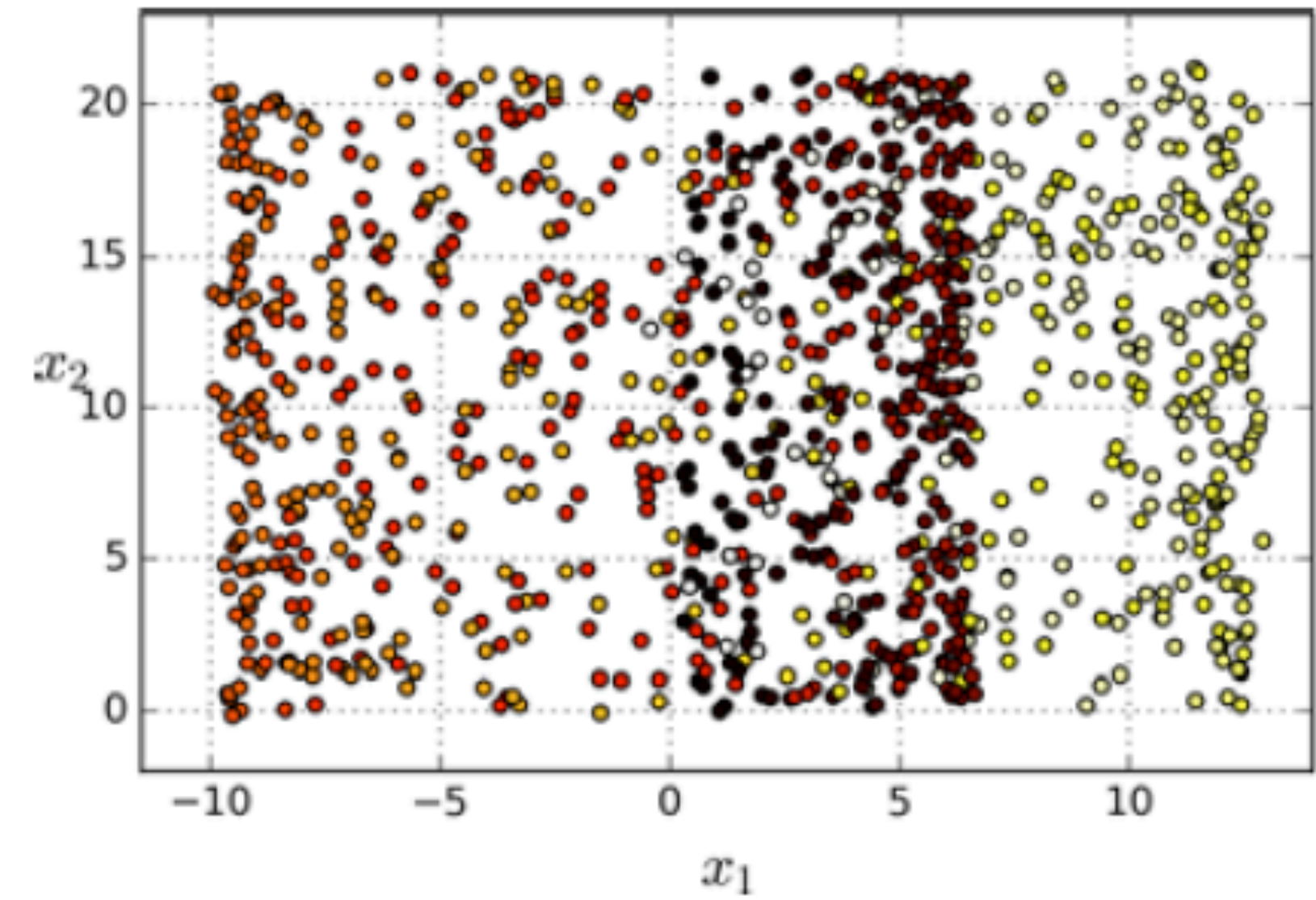




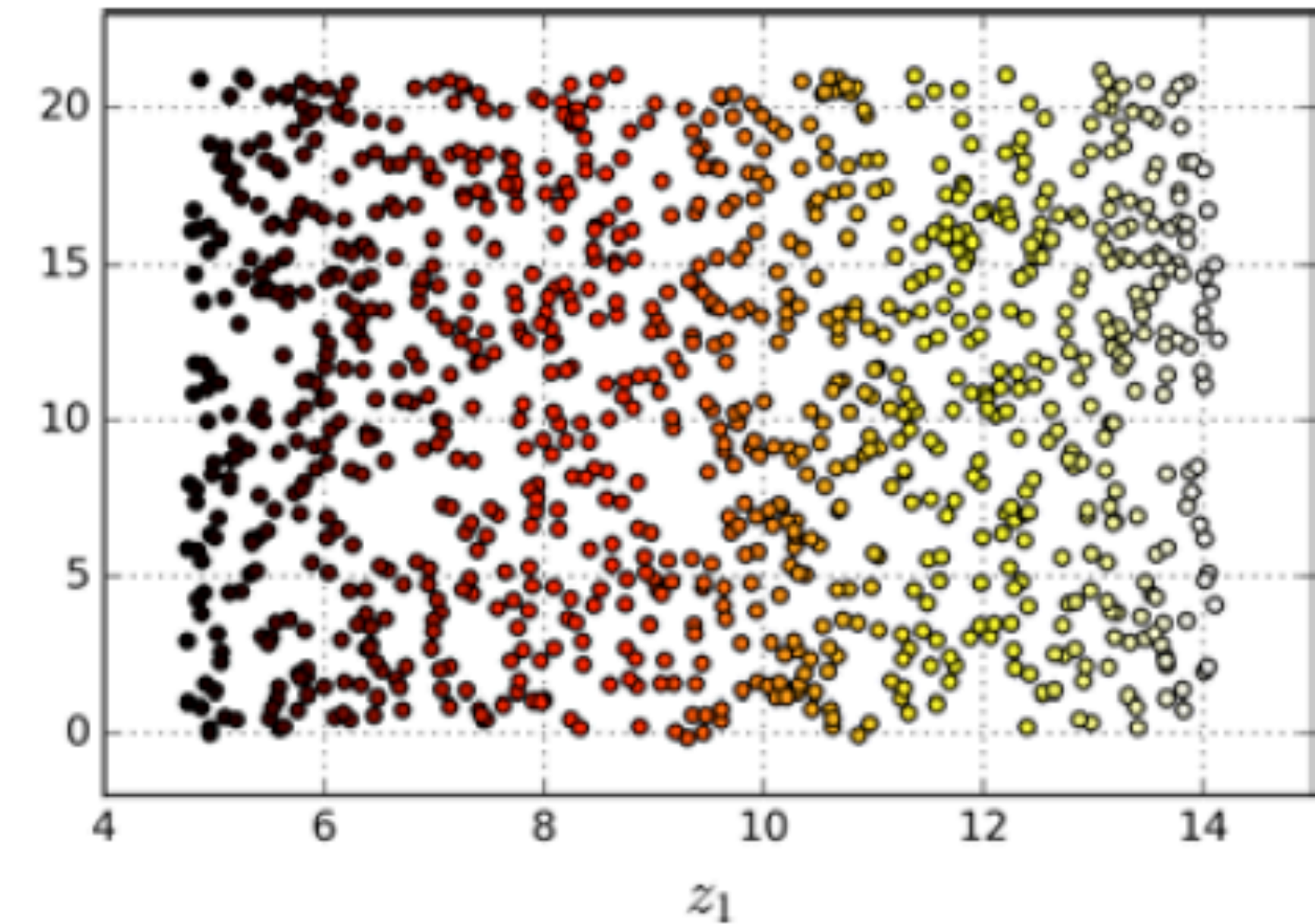
# 스위스롤 (swiss roll)



투영



매니폴드 학습

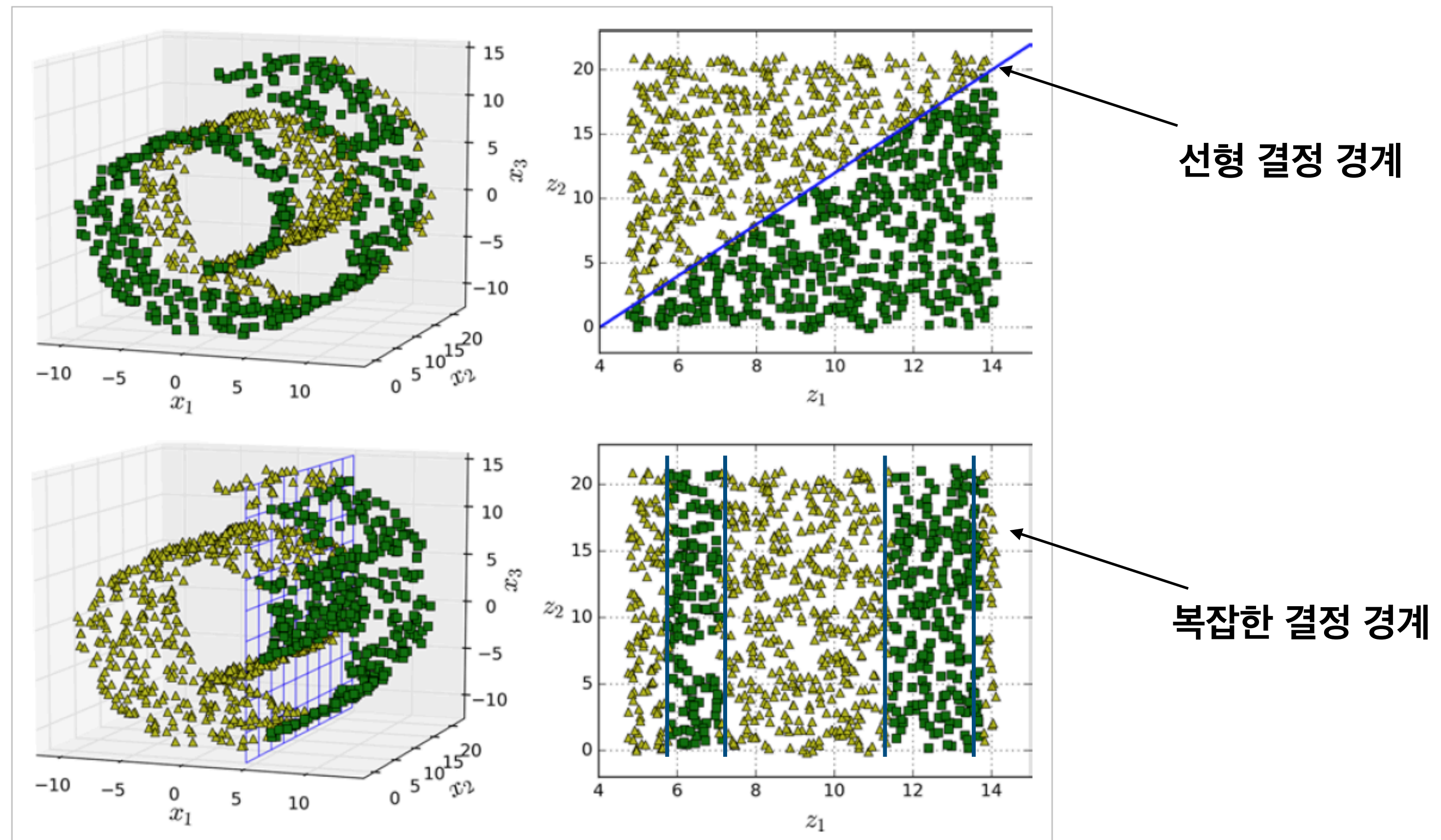




# 매니폴드 학습

- 매니폴드(manifold)는 고차원 공간에서 휘어지거나 뒤틀린 2D 모양을 말합니다.  $d$  차원 매니폴드는 더 높은  $n$  차원 공간에서  $d$  차원 초평면으로 볼 수 있습니다.

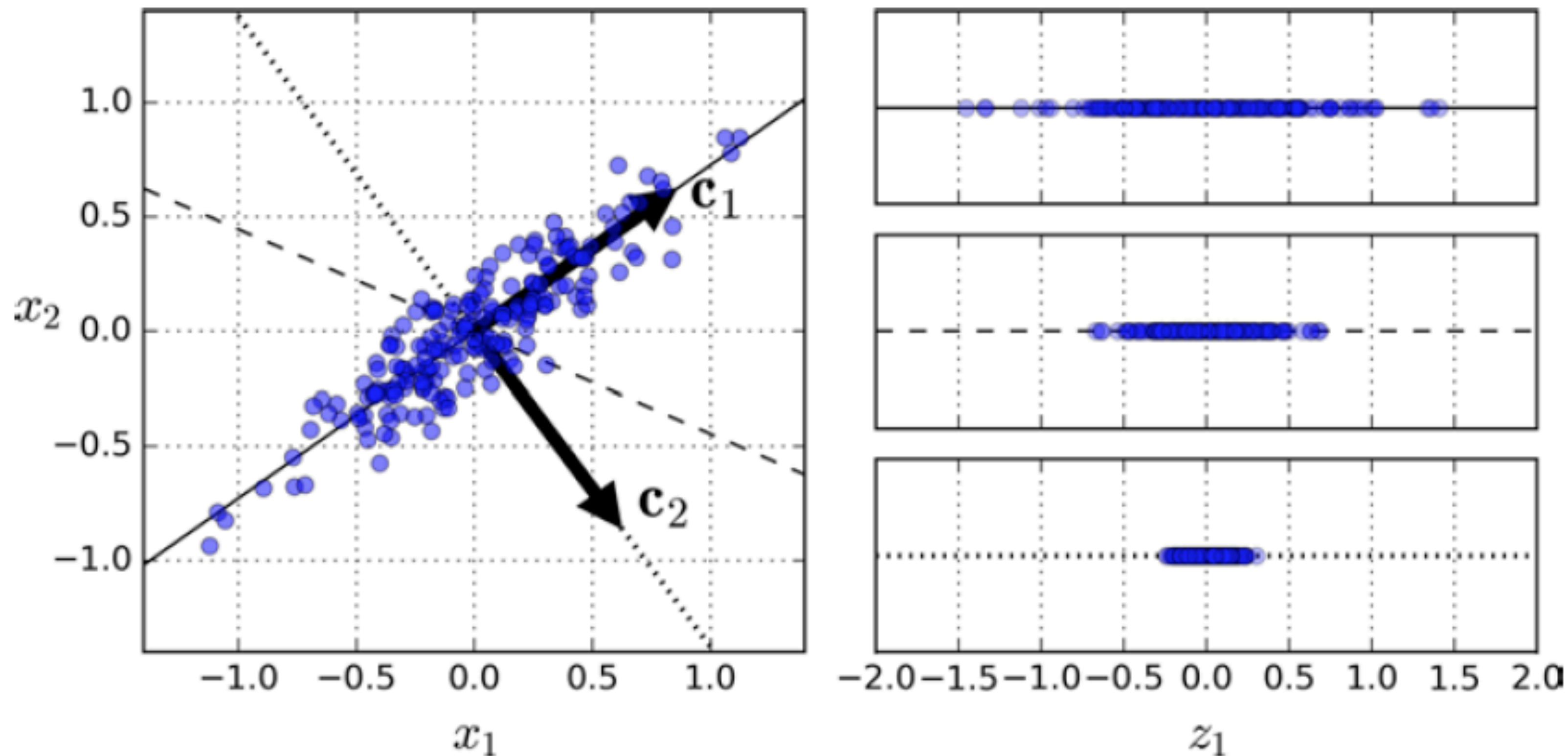
매니폴드 가정이 항상  
성립하는 것은 아닙니다.





# PCA

- 주성분 분석(Principal Component Analysis): 분산을 최대한 보존하는 초평면에 데이터를 투영합니다(원본 데이터와 투영 사이의 평균 제곱 거리가 최소가 되는 초평면).



# 주성분

- 주성분은 서로 직교하며 데이터셋의 특성 개수만큼 찾을 수 있습니다.

- $X$ 를 주성분  $w$ 에 투영했을 때 분산은

$$\mathbf{Var}(Xw) = \frac{1}{n-1}(Xw)^T Xw = \frac{1}{n-1}w^T X^T Xw = w^T \frac{1}{n-1} X^T Xw = w^T Cw$$

평균이 0이라고 가정합니다

- 공분산 행렬의 가장 큰 고유값을 찾으려면

$$C = \frac{1}{n-1}X^T X = \frac{1}{n-1}(U\Sigma V^T)^T(U\Sigma V^T) = \frac{1}{n-1}(V\Sigma U^T)(U\Sigma V^T) = \frac{1}{n-1}V\Sigma^2 V^T = V \frac{\Sigma^2}{n-1} V^T$$

- 즉 특잇값 분해(SVD)에서 구한  $V$ 가 주성분입니다.

$$V = \begin{pmatrix} | & | & & | \\ \mathbf{c}_1 & \mathbf{c}_2 & \cdots & \mathbf{c}_n \\ | & | & & | \end{pmatrix}$$

# 직접 주성분 구해서 투영하기

```
X_centered = X - X.mean(axis=0)  
U, s, Vt = np.linalg.svd(X_centered)
```

svd() 함수는  $V^T$ 를 반환합니다

두 개의 주성분

```
c1 = Vt.T[:, 0]
```

```
c2 = Vt.T[:, 1]
```

$$X_{proj} = X \cdot V_d$$

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

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



# 사이킷런의 PCA

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

```
pca = PCA(n_components = 2)
```

```
X2D = pca.fit_transform(X)
```

← 자동으로 중앙에 맞추습니다.

(60, 2)

(60, 3)

`pca.components_` 가  $V^T$  입니다.

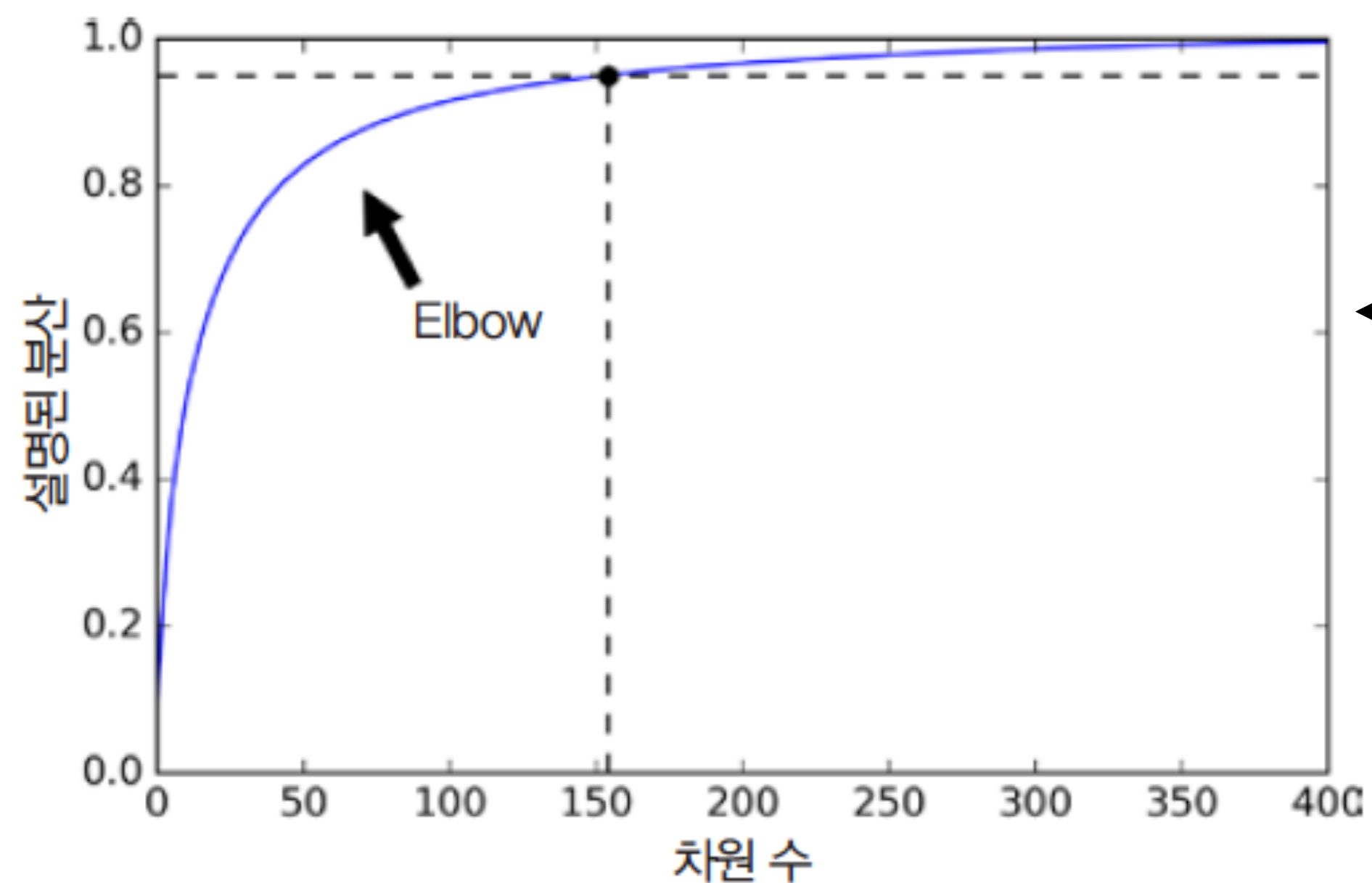
# 설명된 분산의 비율

- 설명된 분산의 비율(explained variance ratio): 주성분의 축을 따라 있는 데이터셋의 분산 비율

$$C = V \frac{\Sigma^2}{n-1} V^T \text{ 에서 고윳값 } \frac{\Sigma^2}{n-1}$$

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

# 차원수 선택



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

```
pca = PCA(n_components=0.95)  
X_reduced = pca.fit_transform(X_train)
```



# 재구성 오차

- 재구성 오차 = (원본 데이터 - PCA변환 뒤 복원한 것)<sup>2</sup>

$$X_{proj} = X \cdot V_d \quad X_{recov} = X_{proj} \cdot V_d^T$$

(154, 784)

95%의 설명된 분산

(52500, 154)

(52500, 784)

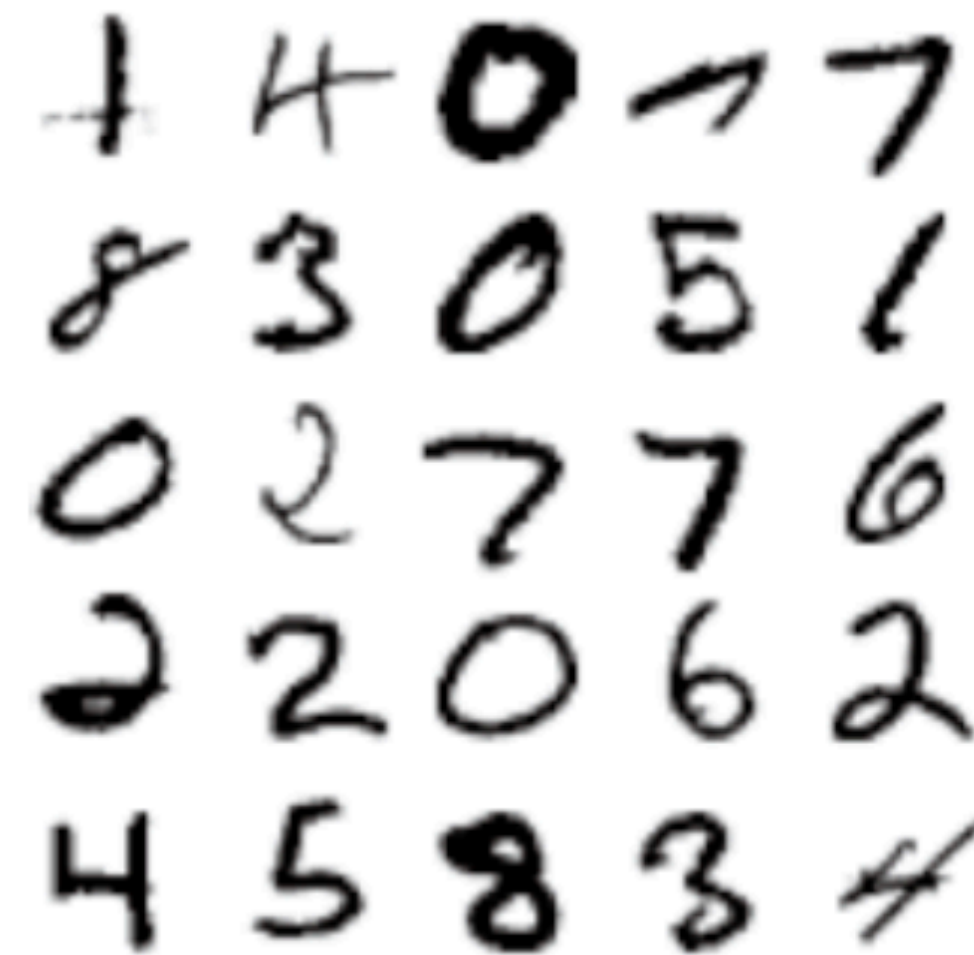
```
pca = PCA(n_components = 154)
```

```
X_reduced = pca.fit_transform(X_train)
```

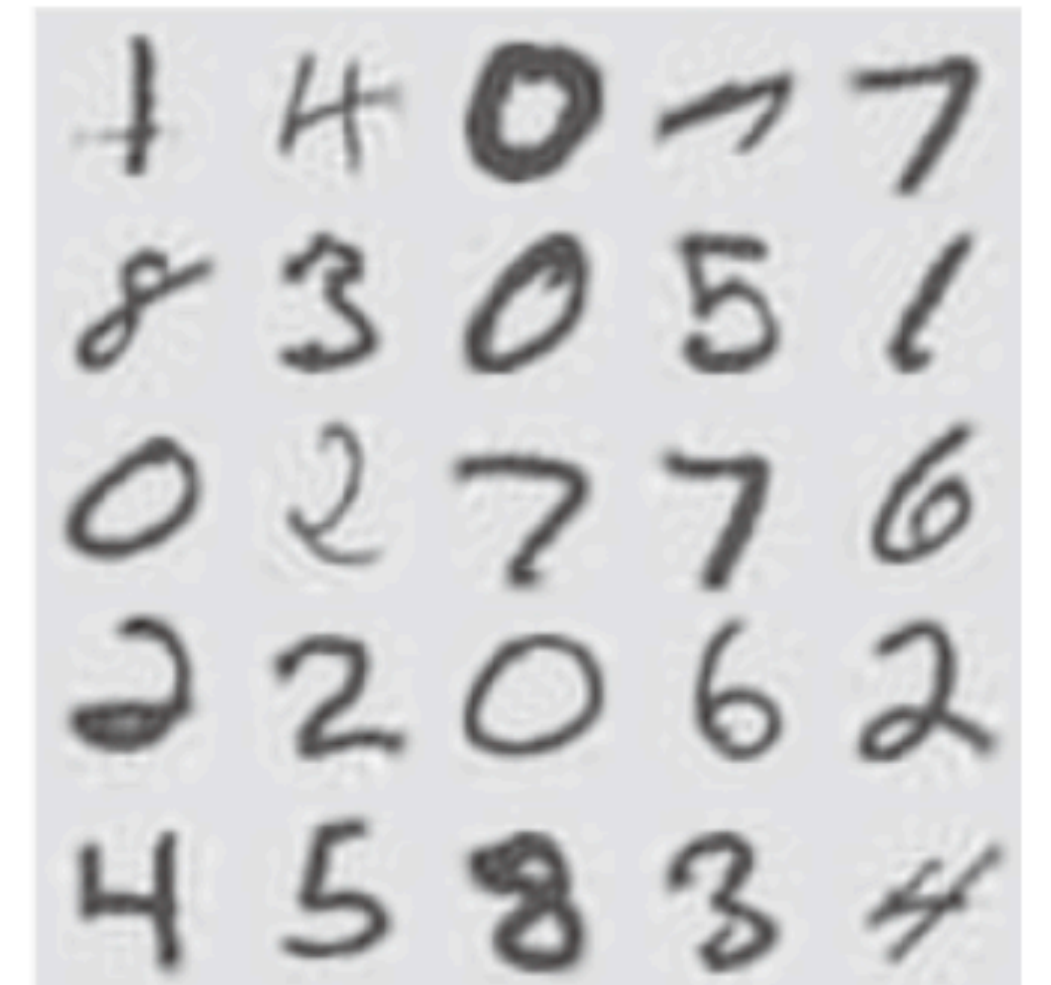
```
X_recovered = pca.inverse_transform(X_reduced)
```

(52500, 784)

원본



압축 후 복원



# 점진적 PCA

- SVD를 사용하면 전체 데이터를 메모리에 적재해야 합니다. 대안으로 IPCA(Incremental PCA)를 사용합니다.

```
from sklearn.decomposition import IncrementalPCA
```

```
n_batches = 100
```

```
inc_pca = IncrementalPCA(n_components=154)
```

```
for X_batch in np.array_split(X_train, n_batches):
```

```
    inc_pca.partial_fit(X_batch)
```

← **partial\_fit** 메서드 호출

```
X_reduced = inc_pca.transform(X_train)
```

```
X_mm = np.memmap(filename, dtype="float32", mode="readonly", shape=(m, n))
```

```
batch_size = m // n_batches
```

```
inc_pca = IncrementalPCA(n_components=154, batch_size=batch_size)
```

```
inc_pca.fit(X_mm)
```

← **fit** 메서드는 **batch\_size** 만큼 나누어  
**partial\_fit** 메서드를 사용합니다

# 랜덤 PCA

- PCA의 계산 복잡도(공분산 + 고유벡터):  $O(m \times n^2) + O(n^3)$
  - 랜덤 PCA의 계산 복잡도:  $O(m \times d^2) + O(d^3)$
- 특성 수
- 주성분 수

```
rnd_pca = PCA(n_components=154, svd_solver="randomized")  
X_reduced = rnd_pca.fit_transform(X_train)
```



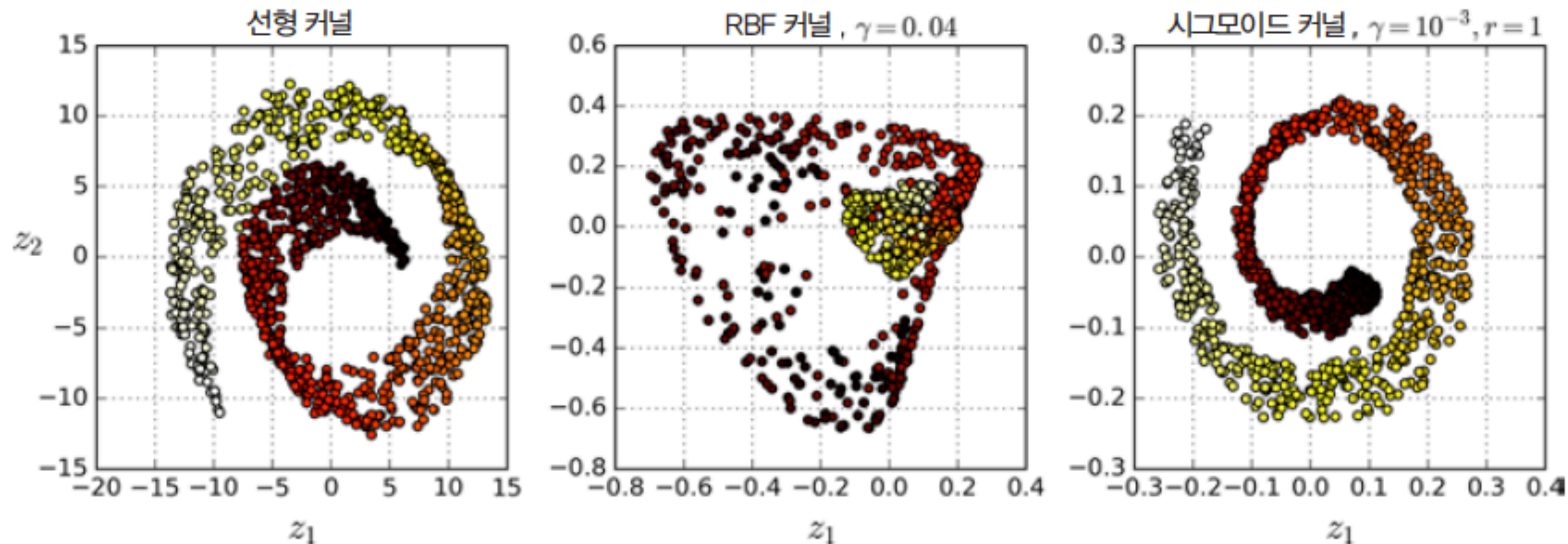
# 커널 PCA

- 서포트 벡터 머신의 커널 트릭을 PCA에 적용해 고차원 공간에서 차원 축소를 하는 효과를 만듭니다.

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

```
rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.04)
```

```
X_reduced = rbf_pca.fit_transform(X)
```



# 커널 PCA 하이퍼파라미터 튜닝

- 차원 축소는 종종 지도 학습의 전처리 단계로 사용됩니다.

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

clf = Pipeline([
    ("kpca", KernelPCA(n_components=2)),
    ("log_reg", LogisticRegression())
])

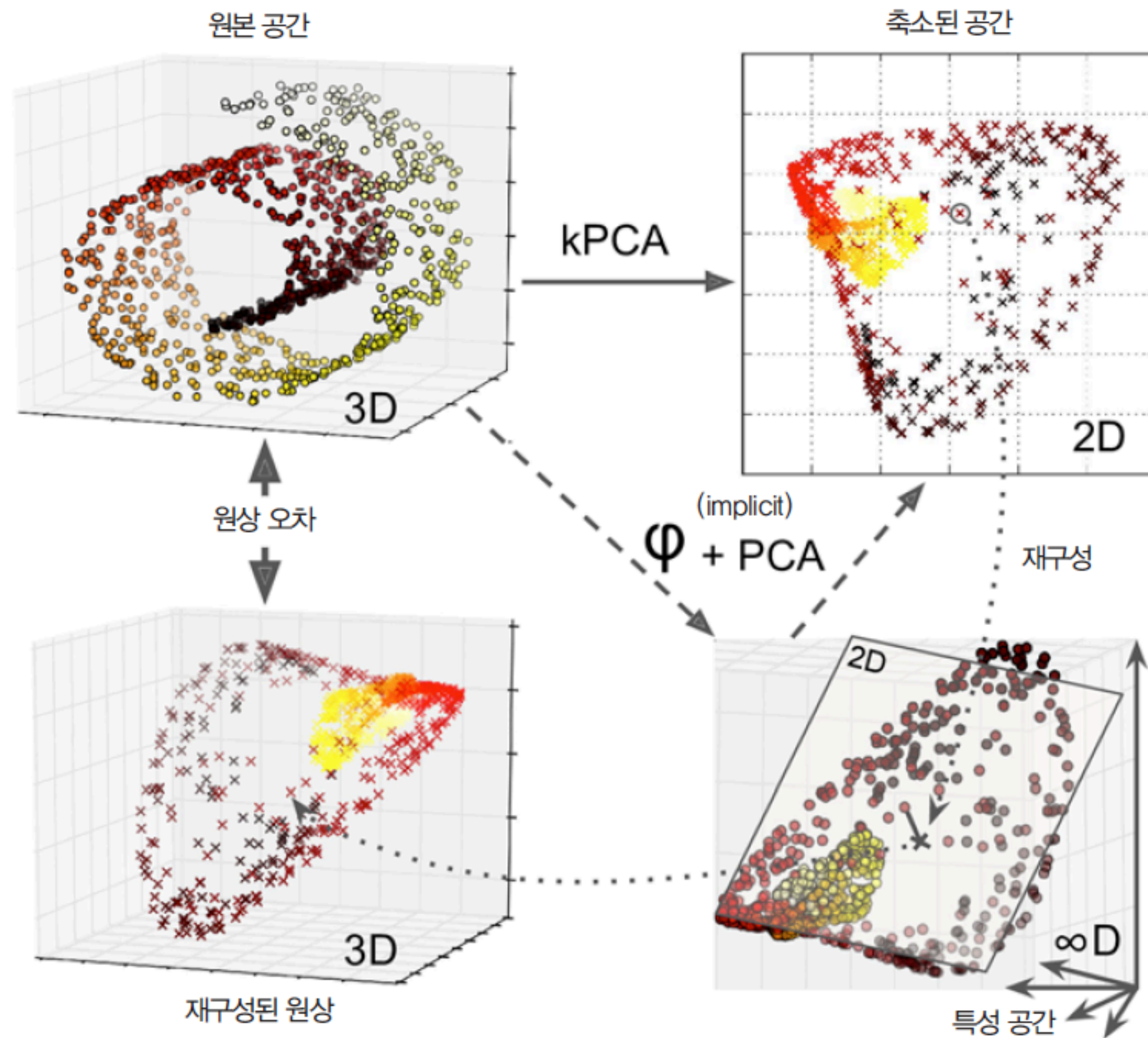
param_grid = [{
    "kpca__gamma": np.linspace(0.03, 0.05, 10),
    "kpca__kernel": ["rbf", "sigmoid"]
}]

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



# 재구성 원상

- 커널 PCA로 투영한 포인트는 커널 트릭이 적용된 고차원 공간으로 복원이 안됩니다.



```
rbf_pca = KernelPCA(n_components = 2, kernel="rbf", gamma=0.0433,  
                    fit_inverse_transform=True)  
X_reduced = rbf_pca.fit_transform(X)  
X_preimage = rbf_pca.inverse_transform(X_reduced)
```

기본값: **False**

```
>>> from sklearn.metrics import mean_squared_error  
>>> mean_squared_error(X, X_preimage)  
32.786308795766132
```



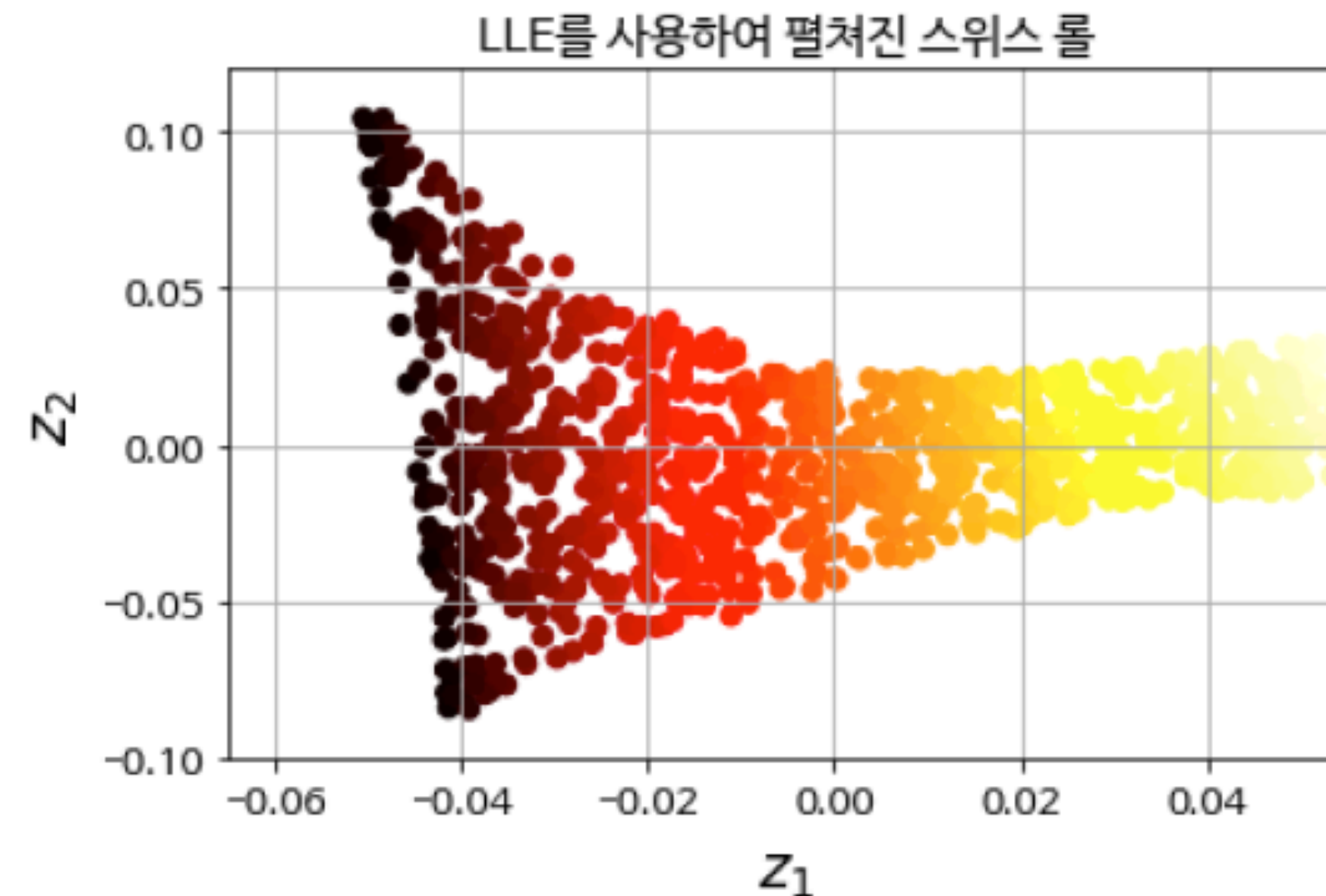
# LLE

- 지역 선형 임베딩(Locally Linear Embedding)은 훈련 샘플이 가까운 이웃과의 관계가 잘 보존되는 저차원 표현을 찾습니다.

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

```
lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10)
```

```
X_reduced = lle.fit_transform(X)
```



# 다른 차원 축소 기법

- 다차원 스케일링(MDS): <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.MDS.html>
- Isomap: <https://scikit-learn.org/stable/modules/generated/sklearn.manifold.Isomap.html>
- t-SNE: <파이썬 라이브러리를 활용한 머신러닝> 3장
- 선형 판별 분석: Sebastian Raschka's Python Machine Learning 2nd Ed.

감사합니다