기계학습_231107

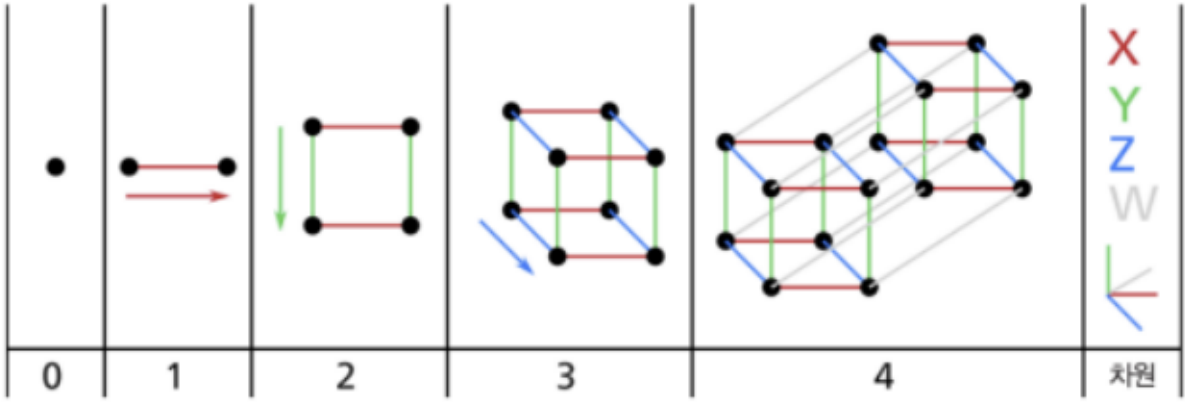
8.0 차원 축소

* 차원의 저주
샘플의 특성이 많으면 학습이 매우 어려워짐.

*차원 축소
- 특성 수를 줄여 학습 불가능한 문제를 학습 가능한 문제로 만드는 기법.
- 훈련 속도 빨라짐 → but 일부 정보의 유실로 성능 저하 위험
- 데이터 시각화에도 유용

8.1 차원의 저주

*고차원(3차원을 초가)은 직관적으로 상상 어려움

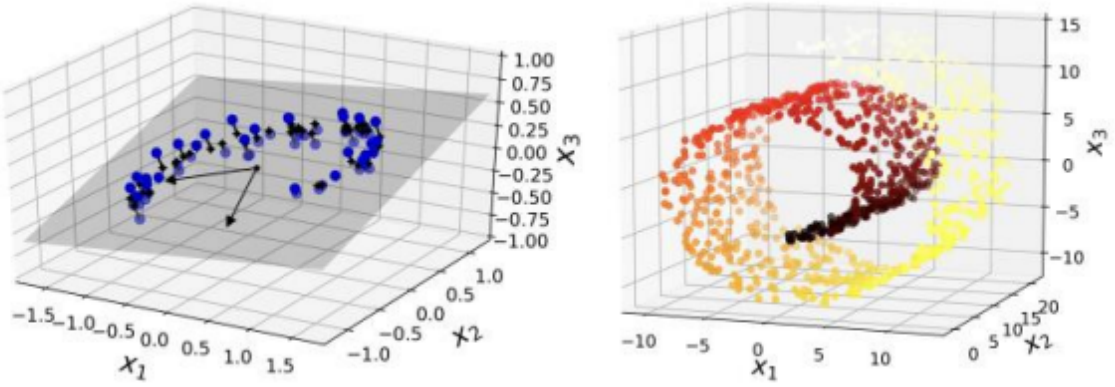


*문제 : 차원 증가 두 지점 사이의 거리가 매우 증가
- 고차원= 많은 공간 가짐, 고차원 데이터셋= 매우희박
- 특성 수가 아주 많은 경우, 과대적합 위험 커짐(훈련샘플 사이 거리가 매우커 기존값을 이용한 추정(예측)이 매우 불안정해짐)
- 해결책 : 샘플 수를 늘리기 → 고차원이라 사실상 불가능

8.2 차원 축소를 위한 접근 방법

*고차원 공간의 데이터 분포
- 대부분의 문제 : 훈련 샘플이 모든 차원에 걸쳐 균일하게 퍼져 있지 X

- 많은 특성= 거의 변화가 X, 다른 특성= 서로 강하게 연관
- 고차원 데이터 분포 예

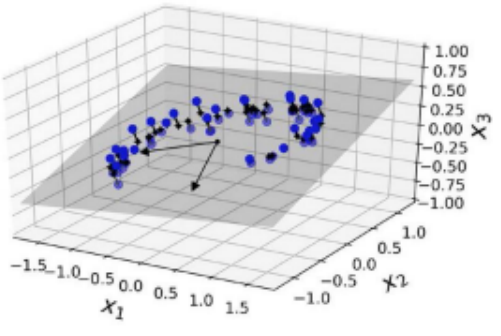


*차원을 감소 시키는 접근법
- 투영 (Projection)
- 매니폴드 학습

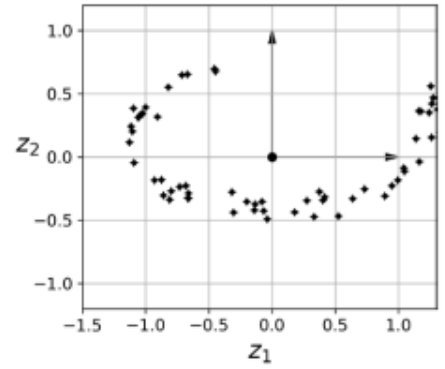
8.2.1 투영

*투영
- n차원 공간에 존재하는 d차원 부분공간을 d차원 공간으로 투영. (단, $d < n$)

*투영 예제
- 모든 훈련 샘플 부분 공간에 수직 투영시 하단 그림 얻음
- 데이터셋의 차원을 3D에서 2D로 줄임.
- 각 축은 새로운 특성 z1과 z2에(평면에 투영된 좌표) 대응



▲ 그림 8-2 2차원에 가깝게 배치된 3차원 데이터셋



▲ 그림 8-3 투영하여 만들어진 새로운 2D 데이터셋

8.2.1 투영 (문제 스위스 롤 데이터셋)

*차원 축소 방법 중 투영이 언제나 최선의 방법은 아님.
- ex) 스위스 롤 데이터셋:부분 공간이 뒤틀림, 휘어 있기도 함.
- 그냥 평면에 투영 : (그림의 왼쪽)스위스 롤의 층이 서로 뭉개짐
- 스위스 롤을 펼침 : (그림의 오른쪽) 2차원 데이터 셋을 얻음

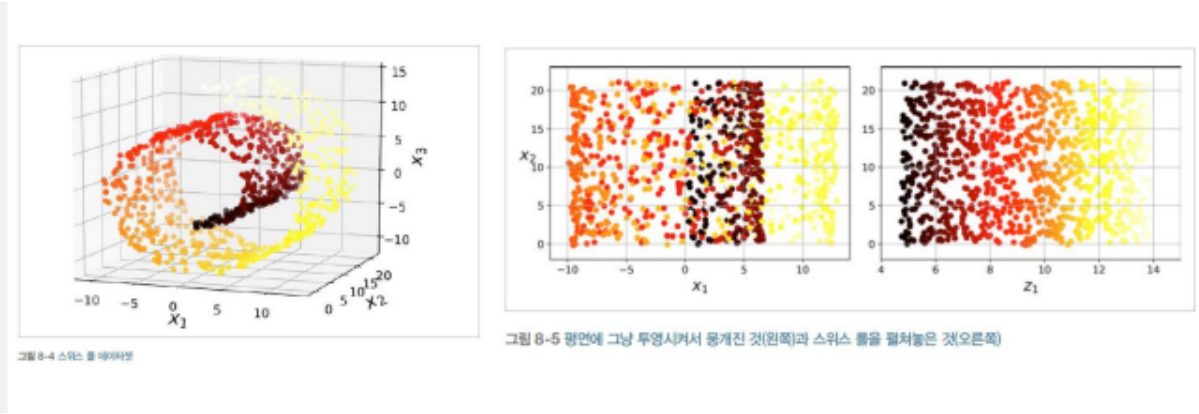
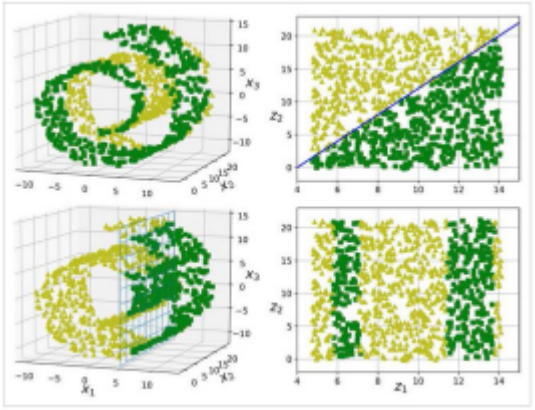


그림 8-5 평면에 그냥 투영시켜서 뭉개진 것(왼쪽)과 스위스 롤을 펼쳐놓은 것(오른쪽)

8.2.2 매니폴드 학습

*매니폴드 학습
- 2D 매니폴드 = 고차원 공간에서 휘어지거나 뒤틀린 2D 모양(ex)스위스 롤)
- d 차원 매니폴드 = 국부적으로 d 차원 초평면으로 보일 수 있는 n 차원 공간의 일부 (d<n)
- 대부분 실제 고차원 데이터셋dl 더 낮은 저차원 매니폴드에 가깝게 놓여 있다는 매니폴드 가정에 근거

*매니폴드 가정의 문제점
- 저차원의 매니폴드 공간으로 차원축소 = 간단한 매니폴드가 됨(가정) → 가정이 항상 유효하지 않음. (훈련속도는 향상)



▲ 그림 8-6 저차원에서 항상 간단하지 않은 결정 경계

8.3 PCA

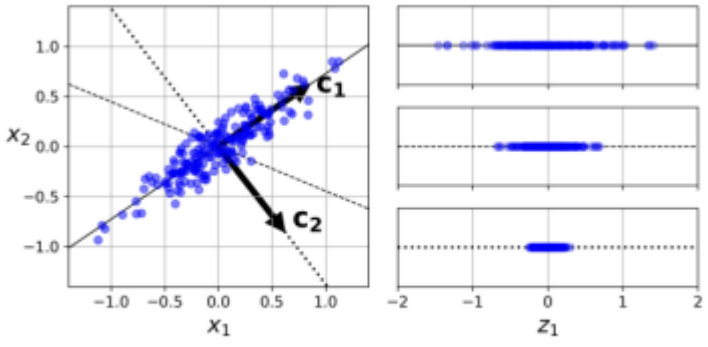
*차원축소 기법 활용 학습 알고리즘
- 투영 기법 알고리즘 : PCA(주성분분석), 커널 PCA(비선형 투영)
- 매니폴드 기법 알고리즘 : LLE(지역 선형 임베딩)

*PCA (Principal Component Analysis)
- 주성분 분석 : 1.데이터에 가장 가까운 초평면 정의 2.데이터를 평면에 투영.
- 분산 보존 개념, 주성분 개념 중요.

8.3.1 분산 보존

*분산 보존
- 저차원으로 투영시 :훈련 세트의 분산이 최대로 보존(정보가장적게 손실)되는 축을 선택
- 원본 데이터셋과 투영된 데이터셋 사이의 평균제곱거리 최소화하는 축

*저차원의 초평면에 훈련 세트를 투영전에 올바른 초평면 선택
- $c1$ 벡터가 위치한 실선 축으로 투영 = 분산을 최대한 보존
- $c2$ 벡터가 위치한 점선 축으로 투영 = 분산을 매우 적게 보존
- 가운데 파선의 투영 = 분산을 중간 정도로 보존



▲ 그림 8-7 투영할 부분 공간 선택하기

8.3.2 주성분

*주성분(PC Principal Component) 축 찾기
- 첫 번째 주성분: 훈련 세트에서 분산을 최대한 보존하는 축
,그림의 실선, 첫 번째 PC = 벡터 C1이 놓인 축.
- 두 번째 주성분: 첫 번째 주성분과 수직을 이룸 = 분산을 최대한 보존하는 축
- 그림에서는 점선, 두 번째 PC는 벡터 C2가 놓인 축.

세 번째 주성분: 첫번째, 두번째 주성분과 수직을 이루면서 분산을 최대한 보존하는 축.
- 데이터 셋에 있는 차원의 수만큼 네 번째, 다섯 번째 … n 번째 축을 찾기
*훈련 셋트의 주성분 찾기
- SVD(Singular Value Decomposition – 특잇값 분해) 사용
: svd() 함수로 모든 주성분을 구하기 → 처음 두 개의 PC를 정의하는 두 개의 단위 벡터 추출

식 8-1 주성분 행렬

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

$X = U \sum V^T$ 에서 V가 주성분

- (m, m) (m, n) (n, n)

m: 샘플 개수, n: 특성 개수

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

8.3.3 d차원으로 투영하기

*주성분 추출후 처음 d 개의 주성분으로 정의한 초평면에 투영하여 데이터셋의 차원을 d 차원으로 축소
- 행렬의 곱셈으로 바로 해결됨.

식 8-2 훈련 세트를 d차원으로 투영하기

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

*파이썬 코드 : 첫 두개의 주성분으로 정의된 평면에 훈련 세트를 투영

```
W2 = Vt.T[:, :2]
X2D = X_centered.dot(W2)
```

8.3.4 사이킷런 사용하기

*사이킷런의 PCA 모델 제공
- SVD 분해 방법 사용하여 구현 : PCA 모델을 사용해 데이터셋의 차원을 2로 줄이기

```
from sklearn.decomposition import PCA

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

PCA 객체를 사용하여 계산된 주성분을 참조할 수 있습니다:

```
pca.components_
```

```
array([[-0.93636116, -0.29854881, -0.18465208],
       [ 0.34027485, -0.90119108, -0.2684542 ]])
```

SVD 방법으로 계산된 처음 두 개의 주성분과 비교해 보겠습니다:

```
Vt[:2]
```

```
array([[ 0.93636116,  0.29854881,  0.18465208],
       [-0.34027485,  0.90119108,  0.2684542 ]])
```

축이 뒤집힌 것을 알 수 있습니다.

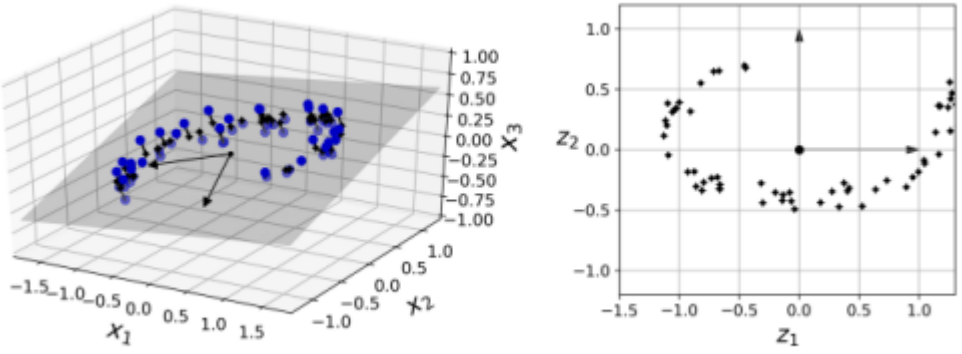
8.3.5 설명된 분산의 비율

*explained_variance_ration_ 속성 변수
- 비율 : 각 주성분의 축에 있는 데이터셋의 분산 비율

```
pca.explained_variance_ratio_
```

```
array([0.84248607, 0.14631839])
```

*아래 투영 그림에서 설명된 3차원 데이터셋
- Z1축 (첫 번째 PC) : 84.2 %
- Z2축 (두 번째 PC) : 14.6 %
- 마지막 주성분에 1.2% 정도만 분산에 기여 = 매우 적은 양의 정보가 포함



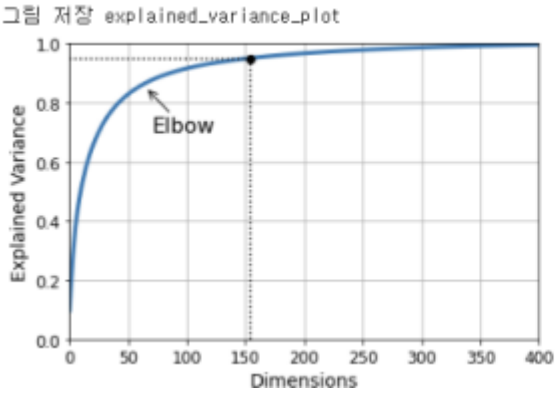
8.3.6 적절한 차원 수 선택하기

*적절한 차원 수
- 차원의 수 : 설명된 분산 비율의 합 = 충분한 분산(95% 정도)
- 데이터 시각화 목적 차원 축소 : 차원을 2,3개로 줄임
*코드
- 차원을 축소X한 PCA를 계산 → 훈련 세트의 분산을 95%로 유지하는 데 필요한 최소한의 차원수 계산

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

```
d
154
```

*설명된 분산을 차원 수에 대한 함수로 그리는 것
- 설명된 분산의 비율의 합의 증가가 완만하게 변하는 지점(Elbow) 선택 가능.



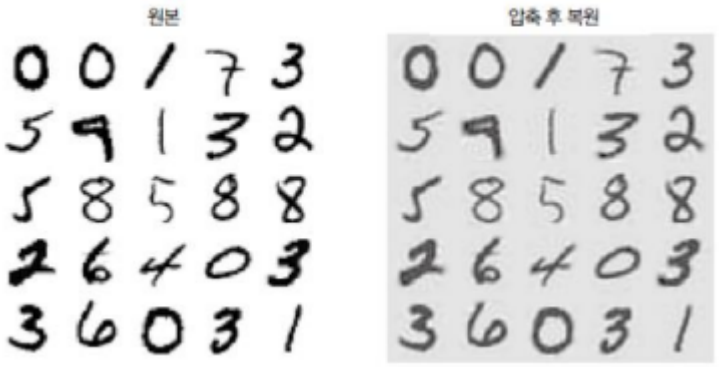
▲ 그림8-8 차원 수에 대한 함수로 나타

8.3.7 압축을 위한 PCA

*차원을 축소 후 = 훈련 세트의 크기가 줄어듦.
- PCA를 MNIST 데이터셋의 차원축소를 위해 사용할 수 있음
*MNIST 데이터셋
- 각 샘플 특성 = 784(28x28)
- 95% 정도의 분산을 유지위해 150개 정도의 주성분만 사용해도 됨
*코드
- MNIST 데이터셋을 154차원으로 압축하고, inverse_transform() 함수를 사용하여 784 차원으로 복원

```
pca = PCA(n_components=154)
X_reduced = pca.fit_transform(X_train)
X_recovered = pca.inverse_transform(X_reduced)
```

*차원 축소 결과
- 784차원→ 154 차원
- 유실된 정보: 5%
- 크기: 원본 데이터셋 크기의 20%
*원본과의 비교
- 정보손실 크지 않음



8.3.8 랜덤 PCA

*랜덤 PCA(확률적 알고리즘) : 처음 d개의 주성분에 대한 근삿값 빠르게 찾음.
*d 가 n 보다 많이 작으면 완전 SVD보다 훨씬 빠름.
*코드
- svd_solver 매개변수를 “randomized”로 지정

```
rnd_pca = PCA(n_components=154, svd_solver="randomized", random_state=42)
X_reduced = rnd_pca.fit_transform(X_train)
```

8.3.9 점진적 PCA

*훈련세트= 미니배치로 나눔 →IPCA에 하나씩 주입 가능.
*온라인 학습에 적용 가능
*코드

- 넘파이의 array_split() 함수 활용
- fit() 메서드(전체훈련 샘플)가 아니라 partial_fit() 메서드를 미니배치마다 호출

```
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):
    print(".", end="") # 책에는 없음
    inc_pca.partial_fit(X_batch)

X_reduced = inc_pca.transform(X_train)
```

*넘파이의 memmap() 클래스 활용 가능
- 바이너리 파일로 저장된 (매우 큰) 데이터셋을 마치 메모리에 들어있는 것처럼 취급할 수 있는 도구 제공
- 미니배치/온라인 학습 가능

8.4 커널 PCA

*커널트릭을 PCA 적용, 차원 축소를 위한 복잡한 비선형 투영을 수행
*투영된 후에 샘플의 군집을 유지, 매니폴드에 가까운 데이터 셋을 펼칠때 유용함.
*코드
- 사이킷런의 KernelPCA를 사용해 RBF 커널로 kPCA를 적용

```
from sklearn.decomposition import KernelPCA

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

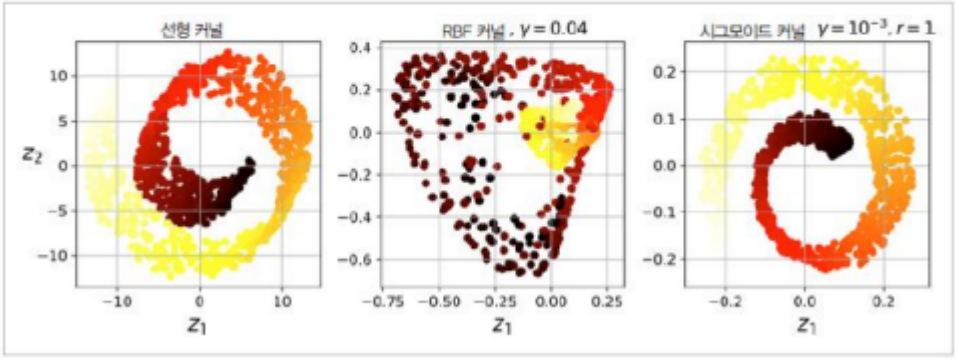


그림 8-10 여러 가지 커널의 kPCA를 사용해 2D로 축소시킨 스위스 롤

8.4.1 커널선택과 하이퍼파라미터 튜닝

*kPCA는 비지도 학습
- 좋은 커널, 하이퍼파라미터를 선택 위한 명확한 성능 측정 기준 X
*커널과 하이퍼파라미터 튜닝을 위한 측정 방식
- 방식 1 : 전처리 용도로 사용 후 예측기와 연동하는 그리드탐색 등을 활용 성능 측정가능

```
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(solver="lbfgs"))
    ])

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)
```

8.4.1 커널선택과 하이퍼파라미터 튜닝

*커널과 하이퍼파라미터 튜닝을 위한 측정 방식
- 방식 2 : 가장 낮은 재구성 원상의 오차를 최소화하는 커널과 하이퍼파라미터 선택 가능
(재구성 원상 – 재구성된 포인트에 가깝게 매핑된 원본 공간의 포인트를 찾을 수 있는 것.)
- 사이킷런에서 fit_inverse_transform=True로 지정하면 재구성 수행함

```
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)
```

```
from sklearn.metrics import mean_squared_error

mean_squared_error(X, X_preimage)
```

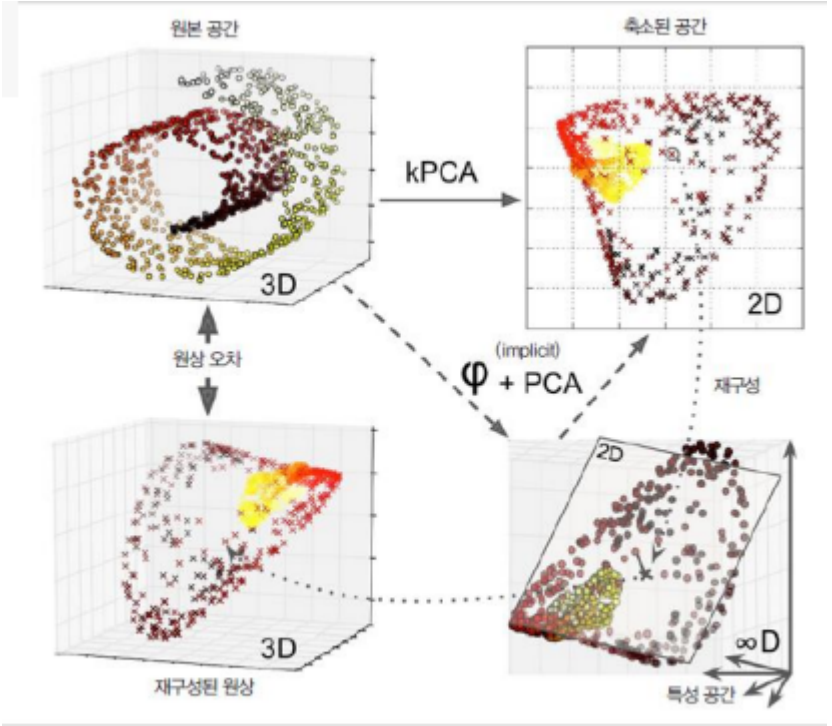


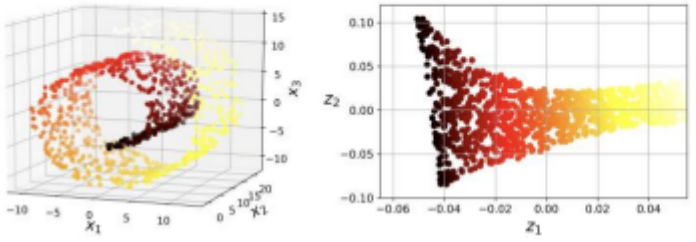
그림 8-11 커널 PCA와 재구성 원상 오차

8.5 LLE

*지역선형임베딩(LLE)은 비선형 차원축소 기법
*투영이 아닌 매이폴드 학습에 의존
*LLE의 Key Idea
- 가장 가까운 이웃에 얼마나 선형적으로 연관되어 있는지 측정
- 국부적인 관계가 가장 잘 보존되는 훈련 세트의 저차원 표현을 찾음.
- 잡음이 너무 많지 않을시 꼬인 매니폴드를 펼침

```
from sklearn.manifold import LocallyLinearEmbedding

lle = LocallyLinearEmbedding(n_components=2, n_neighbors=10, random_state=42)
X_reduced = lle.fit_transform(X)
```



▲ 그림 LLE를 사용하여 펼쳐진 스위스 롤

8.6 다른 차원 축소 기법

* 이전 학습랜덤 투영(random projection)
랜덤한 선형 투영을 사용해 데이터를 저차원 공간으로 투영
*다차원 스케일링(MDS, multidimensional scaling)
샘플 간의 거리를 보존하면서 차원을 축소
*Isomap
각 샘플을 가장 가까운 이웃과 연결하는 식으로 그래프를 만들고, 샘플 간의 지오데식 거리(geodesic distance)를 유지하며 차원축소
*t-SNE(t-distributed stochastic neighbor embedding)
비슷한 샘플은 가까이, 비슷하지 않은 샘플은 멀리 떨어지도록 하면서 차원을 축소
*선형 판별 분석(LDA, linear discriminant analysis)
분류 알고리즘, 훈련 과정에 클래스 사이를 가장 잘 구분하는 축(데이터가 투영되는 초평면을 정의시 사용)을학습

[차원축소에한 장/단점 범위 생각하기]

```
차원축소가 PCA에서 활용될 수 있는 부분은 장/단점으로 분류할 수 있다

(장점)
- 데이터의 분산을 최대한 보존하면서 차원을 축소
- 계산 효율성이 높고, 구현이 간단
- 데이터 시각화 및 노이즈 제거에 유용
(단점)
- 선형 관계에만 효과적이며, 비선형 구조를 잘 포착하지 못함
- 데이터의 분산만을 고려하기 때문에, 분산이 높지 않은 중요한 정보를 잃을 수 있음

한결: 장점과 단점을 고려해봤을 때 대규모 데이터셋의 차원을 간단하게 축소할 때 유용하다고 생각해
다빈: 그러면 패턴 인식 및 기계 학습에서 특징 추출에 사용될 수 있을거야
```