




---

▷ 차원 축소

# 차원 축소 (Dimension Reduction)



우송대학교 AI SW 과정  
이재성



# 학습 내용

---

- ❖ 차원의 저주: 샘플의 특성이 너무 많으면 학습이 매우 어려워짐.
- ❖ 차원축소: 특성 수를 (크게) 줄여서 학습 불가능한 문제를 학습 가능한 문제로 만드는 기법 차원을 축소하면 정보손실 발생하여 성능 저하 가능. 하지만 학습속도 빨라짐.

# MNIST

---

## ❖ 예제: MNIST

- 사진의 중앙에만 집중해도 숫자 인식에 별 문제 없음.  
주성분분석(PCA) 기법을 이용하여 154개의 픽셀만 대상으로 학습 가능

## ❖ 예제: 데이터 시각화

- 차원 수를 2, 3차원으로 줄이면 그래프 시각화 가능  
군집 같은 시각적인 패턴을 감지하여 데이터에 대한 통찰 얻을 수 있음.

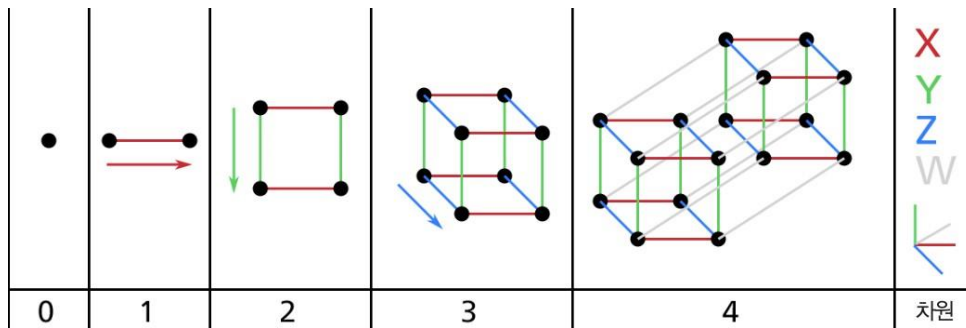
# 학습 내용

---

- ❖ 차원의 저주
- ❖ 차원축소를 위한 접근법 사영 다양체학습
- ❖ PCA(주성분분석): 분산 보존 주성분
- ❖ 커널 PCA
- ❖ LLE(지역 선형임베딩)

# 차원의 저주

- 3차원을 초과하는 고차원의 공간 상상 매우 어려움.



# 차원의 저주

---

- ❖ 차원의 커질 수록 두 지점 사이의 거리가 매우 커짐.
- ❖ 즉, 특성 수가 아주 많은 경우, 훈련 샘플 사이의 거리가 매우 커서 과대 적합 위험도가 커짐. 이유: 두 샘플 사이의 거리가 멀어서 기존 값들을 이용한 추정(예측)이 매우 불안정해지기 때문.
- ❖ 해결책: 샘플 수 늘리기. 하지만 고차원의 경우 충분히 많은 샘플 수를 준비하는 일은 사실상 불가능.

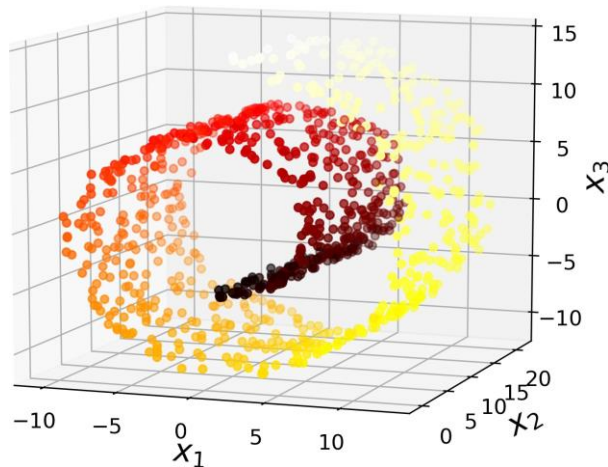
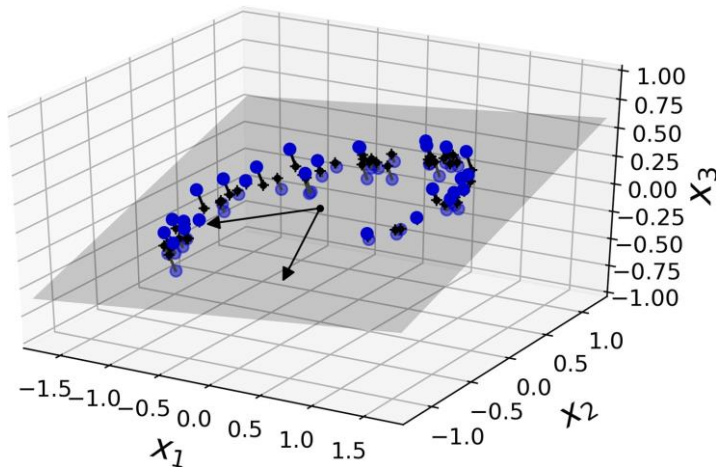
# 차원 축소를 위한 접근법

---

모든 훈련 샘플이 고차원 공간 안의 저차원 부분공간에 가깝게 놓여 있는 경우가 일반적으로 발생

# 예제: 3D to 2D

- ❖ 예제: 아래 두 이미지 모두 3차원(3D) 이미지를 보여줌.
  - 왼편: 2차원(2D) 평면에 가까움.
  - 오른편: 2차원 평면이 휘어진 모양의 롤케이크.
  - 국소적으로 2차원 평면에 가까움.





# 대표적인 차원축소 기법

---

## ❖ 대표적인 차원축소 기법

- 사영(projection)
- 다양체학습(manifold learning)

# 사영(Projection)

---

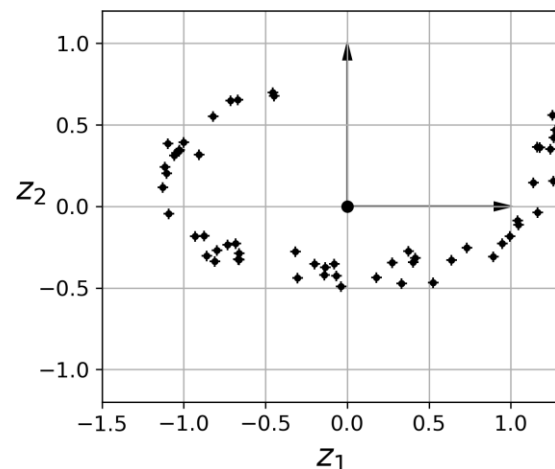
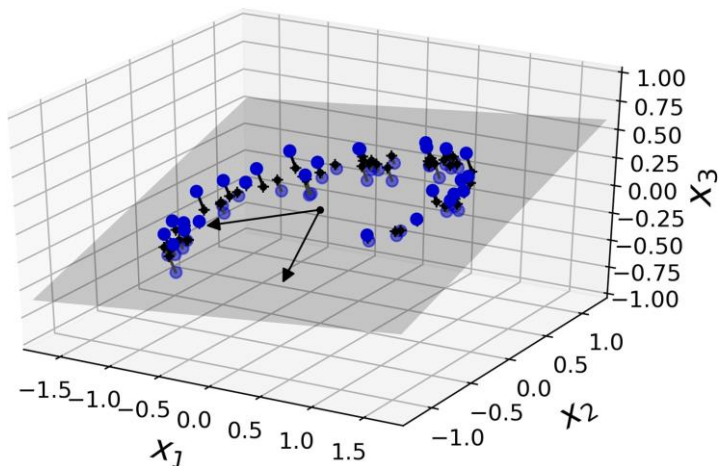
❖  $n$ 차원 공간에 존재하는  $d$ 차원 부분공간을  $d$ 차원 공간으로 사영하기. 단,  $d < n$ .

# 예제: 사영

## 적절한 사영 예제

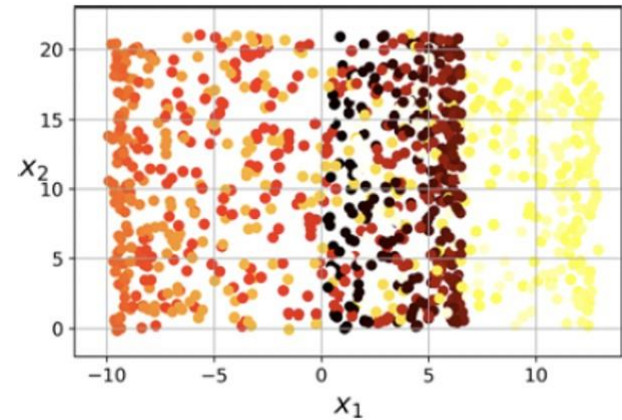
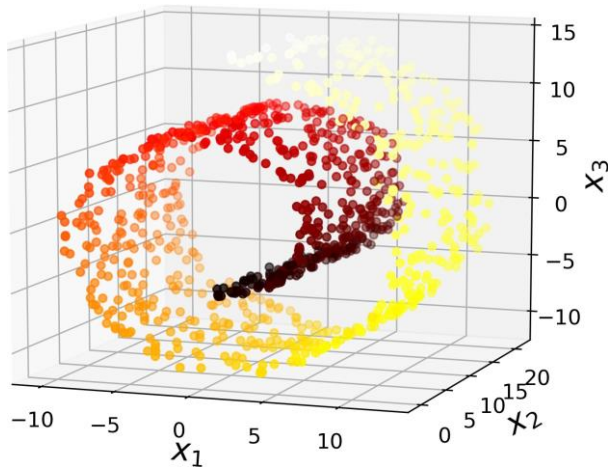
왼쪽 3차원에 존재하는 적절한 2차원 평면으로 사영하면 적절한 2차원 상의 이미지를 얻게됨.

오른쪽 2차원 이미지에 사용된 축  $z_1$  과  $z_2$  를 적절하게 찾는 게 중요 과제임.



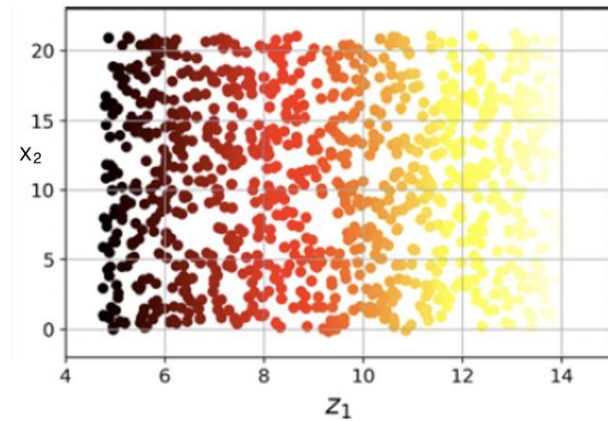
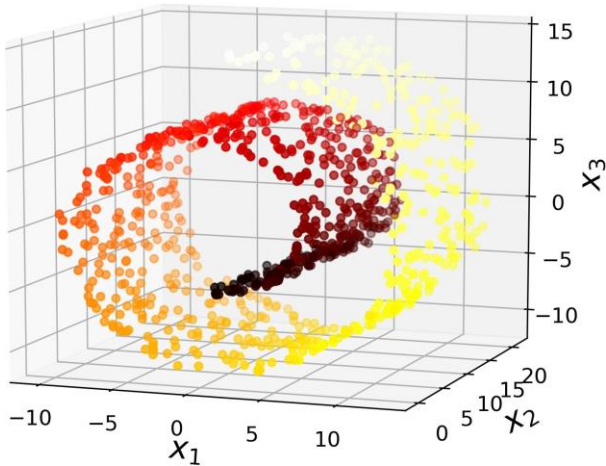
# 부적절한 사영

사영이 경우에 따라 보다 복잡한 결과를 낼 수 있음.  
롤케이크를  $x_1$  과  $x_2$  축으로 사영하면 샘플구분이 보다 어려워짐.



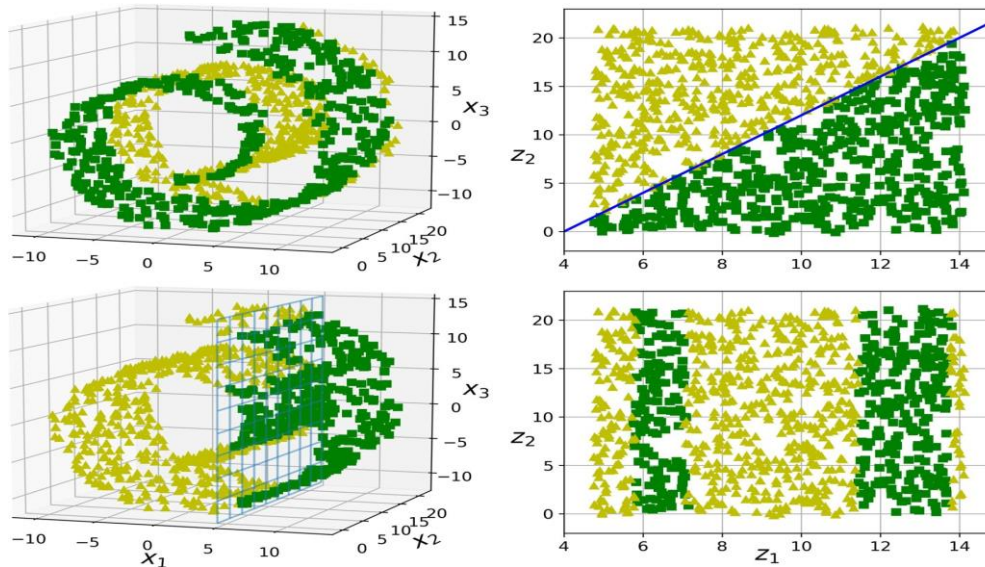
# 다양체 학습

롤케이크의 경우 사영 보다는 롤케이크를 펼치면 보다 적절한 2차원 이미지를 얻게 됨.



# 다양체 가정

- 대부분의 고차원 데이터셋이 더 낮은 차원의 다양체게 가깝다는 가정
- 저차원의 다양체 공간으로 차원축소를 진행하면 보다 간단한 다양체가 된다는 가정과 함께 사용
- 일반적으로 사실 아님. 다음 그림 둘째 행 참조(분포에 따라 다름)



# 차원 축소 기법 활용 알고리즘

---

## ❖ 사영 기법 알고리즘

PCA(주성분분석)  
커널 PCA (비선형 사영)

## ❖ 다양체학습 기법 알고리즘

LLE(지역 선형 임베딩)

# PCA(Principal Components Analysis)

---

- ❖ 훈련 데이터에 가장 가까운 초평면(hyperplane)을 정의한 다음, 그 평면에 사영하는 기법 주성분분석으로 알려짐
- ❖ 분산 보존 개념과 주성분 개념이 중요함.



# 분산보존

---

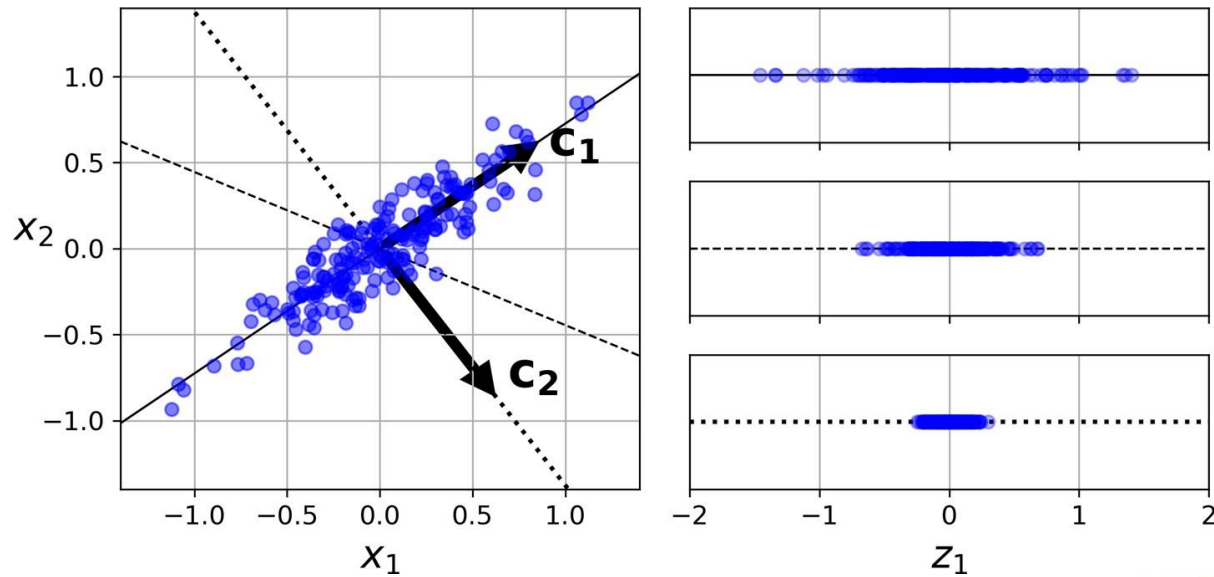
## 분산 보존

저차원으로 사영할 때 훈련 세트의 분산이 최대한 유지되도록 축을 지정해야 함

# 예제

## 예제

아래 그림에서  $c_1$  벡터가 위치한 실선 축으로 사영하는 경우가 분산을 최대한 보존함.



# 축 선택

---

## 축 선택

- 분산을 최대한 보존하는 축을 선택해야 함.
- 원본 데이터셋과 사영된 데이터셋 사이의 평균제곱거리 최소화  
정보손실이 가장 적음.

# 주성분

---

## 주성분

- 첫째 주성분: 분산을 최대한 보존하는 축
- 둘째 주성분: 첫째 주성분과 수직을 이루면서 분산을 최대한 보존하는 축
- 셋째 주성분: 첫째, 둘째 주성분과 수직을 이루면서 분산을 최대한 보존하는 축

...

# 예제

---

## 예제

- $d = 2$ 인 경우: 첫째 주성분이 정해지면 둘째 주성분은 자동으로 선택 (그림에서  $c_2$ 는 자동으로 정해짐)
- 일반적으로  $d$ 차원으로 사영하고자 하는 경우:  $c_1, \dots, c_d$  개의 주성분을 선택해야 함.

# 주성분 찾기

---

## 주성분 찾기

특잇값 분해(SVD) 기법을 이용하면 쉽게 해결됨.

# D 차원 사영

---

$d$ 번째 까지의 주성분을 이용하여 데이터셋을  $d$ 차원으로 사영하는 일은 행렬의 곱셈으로 바로 해결됨.

# 사이킷런 사용

---

❖ 사이킷런의 PCA 모델 제공 SVD 기법 활용

예제: 데이터셋의 차원을 2로 줄이기

```
from sklearn.decomposition import PCA  
pca =  
PCA(n_components = 2)  
X2D = pca.fit_transform(X)
```

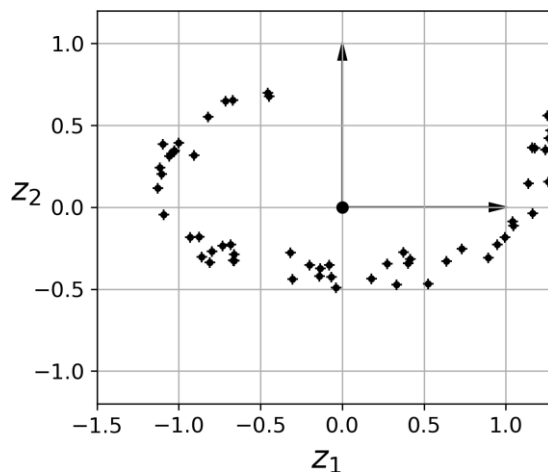
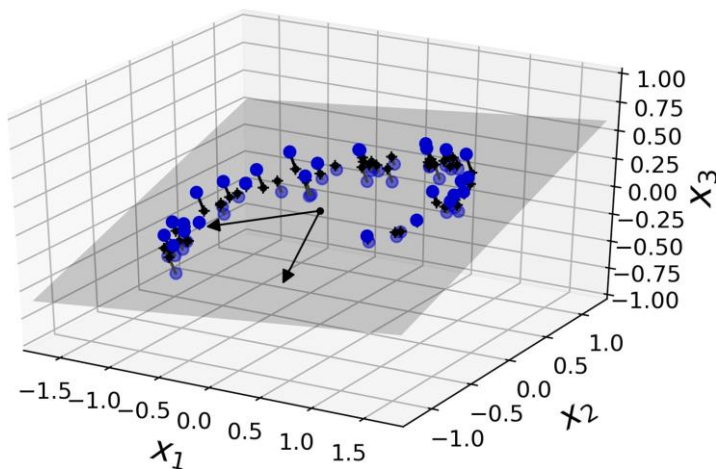


# 분산 비율

explained\_variance\_ratio\_ 속성 변수: 각 주성분에 대한 원 데이터셋의 분산 비율 저장

예제

- 아래 사영 그림에서 설명된 3차원 데이터셋의 경우.
  - $z_1$  축: 84.2%
  - $z_2$  축: 14.6%
- 따라서 마지막 주성분에는 1.2% 정도만 분산에 기여하므로 매우 적은 양의 정보가 포함되어 있음.

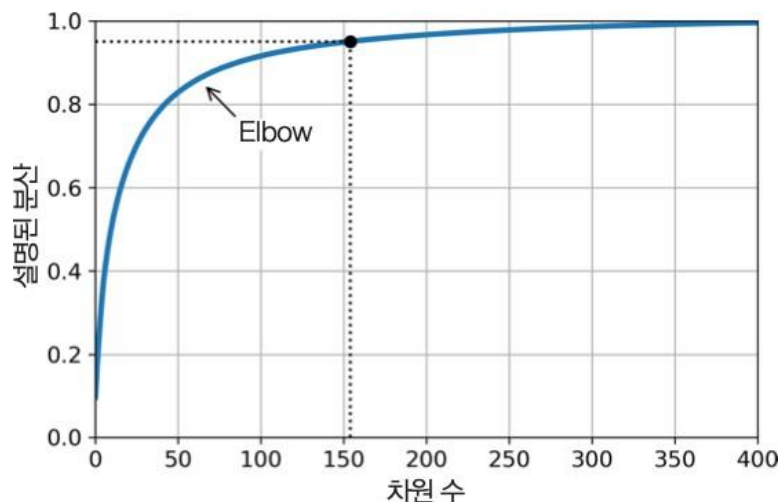


# 적절한 차원 수 선택

추천 적절한 차원 수: 밝혀진 분산 비율의 합이 95% 정도 되도록 하는 주성분들로 구성 단, 데이터 시각화 목적: 2개 또는 3개

## 차원 수 선택 방법 예제

- 밝혀진 분산 비율의 합과 차원 수 사이의 그래프 활용
  - 밝혀진 분산의 비율의 합의 증가가 완만하게 변하는 지점(elbow) 선택 가능
- 아래 그림: 100 차원 선택 가능



# MNIST 활용 예제 압축을 위한 PCA

---

- PCA를 MNIST 데이터셋의 차원축소를 위해 사용할 수 있음.
- MNIST 데이터셋의 주성분분석을 통해 95% 정도의 분산을 유지하려면 150개 정도의 주성분만 사용해도 됨
- 아래 코드: 154개 주성분 사용하여 차원축소하기

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

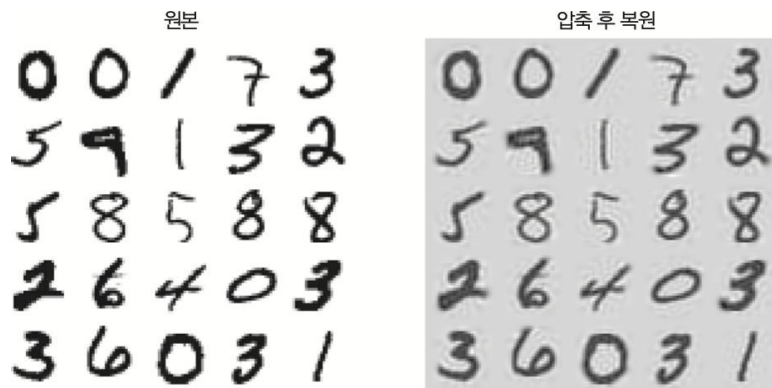
```
X_reduced = pca.fit_transform(X_train)
```

# 차원 축소 결과

차원 축소 결과:

- 784차원을 154 차원으로 줄임.
- 유실된 정보: 5%
- 크기: 원본 데이터셋 크기의 20%

원본과의 비교: 정보손실 크지 않음 확인 가능



# 랜덤 PCA

---

- 주성분 선택을 위해 사용되는 SVD 알고리즘을 확률적으로 작동하도록 만드는 기법 보다 빠르게 지정된 개수의 주성분에 대한 근사값을 찾아줌
- $d$ 가  $n$  보다 많이 작으면 기본 SVD 보다 훨씬 빠름
- 아래 코드: `svd_solver` 옵션을 "randomized"로 설정

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

# 점진적 PCA

- 훈련세트를 미니배치로 나눈 후 IPCA(점진적 PCA)에 하나씩 주입 가능
- 온라인 학습에 적용 가능
- 아래 코드: 넘파이의 `array_split()` 함수 활용

`partial_fit()` 활용에 주의

```
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)

X_reduced = inc_pca.transform(X_train)
```

# 넘파이 memmap()

---

## ❖ 넘파이의 `memmap()` 클래스 활용 가능

- 바이너리 파일로 저장된 (매우 큰) 데이터셋을 마치 메모리에 들어있는 것처럼 취급할 수 있는 도구 제공
- 이를 이용하여 미니배치/온라인 학습 가능

# 커널 PCA

---

- 커널 트릭을 역으로 활용하는 기법.
- 복잡한 비선형 사영 수행.
- 사영된 후에 샘플의 군집을 유지하거나 꼬인 다양체 모양의 데이터셋을 펼칠 때 유용.

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

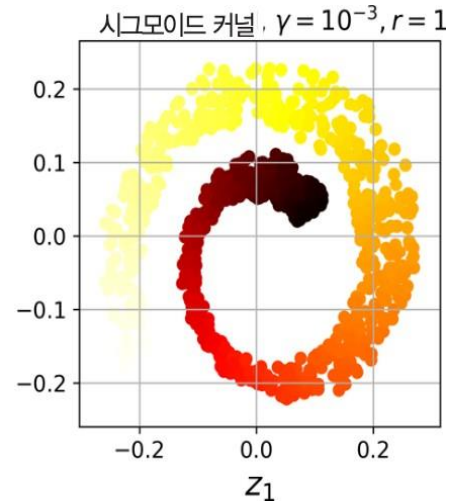
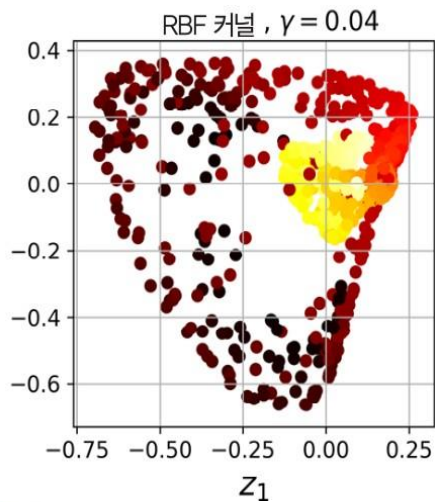
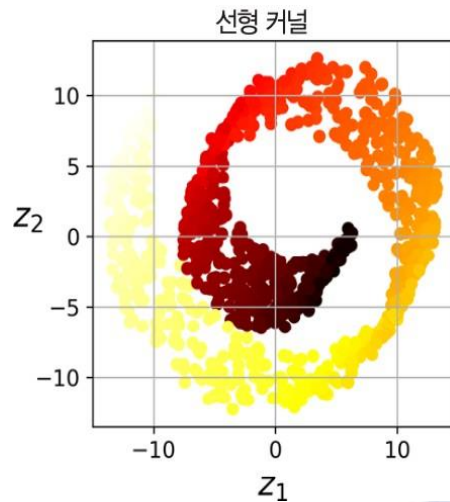
```
rbf_pca = KernelPCA(n_components = 2,  
    kernel="rbf", gamma=0.04)
```

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



# 커널 예제

롤케이크에 적용한 다양한 커널 예제



# 커널과 하이퍼파라미터 튜닝

kPCA는 비지도 학습

비지도 학습은 명확한 성능 측정 기준 없음. 하지만 ...

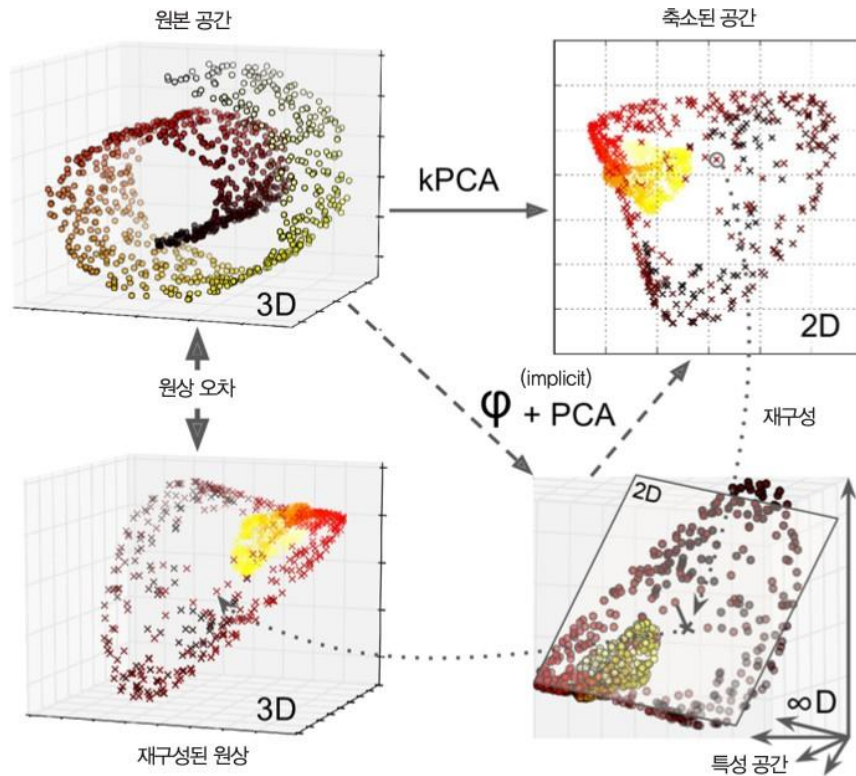
- ❖ 방식 1: 전처리 용도로 사용 후 예측기와 연동하는 그리드탐색 등을 활용하여 성능 측정 가능

```
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"] }]
```

# 커널과 하이퍼파라미터 튜닝

- ❖ 방식 2: 가장 낮은 재구성 오차를 만드는 커널과 하이퍼파라미터 선택 가능  
단점: kPCA의 재구성은 선형 PCA 보다 훨씬 어려움. 아래 그림 참조



# LLE(Local Linear Embedding)

---

지역선형임베딩(LLE)은 비선형 차원축소 기법 사영이 아닌 다양체학습에 의존

## ❖ 기본 아이디어

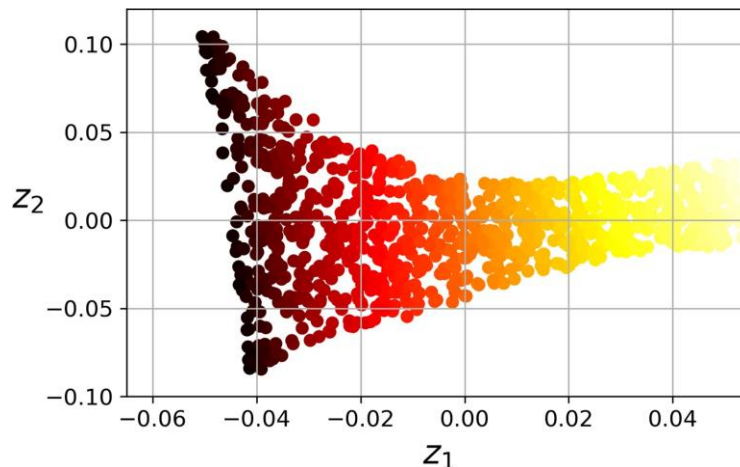
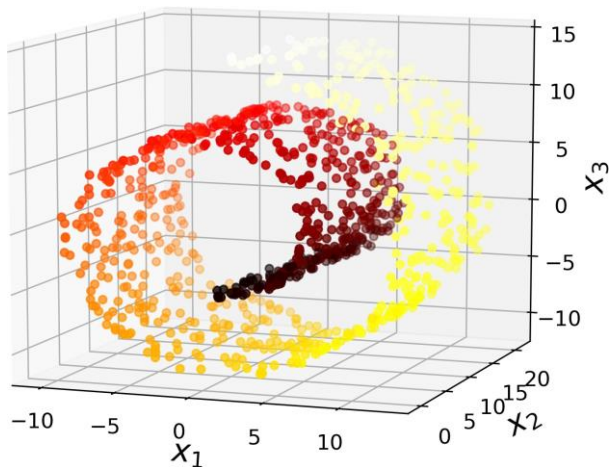
- 가장 가까운 이웃에 얼마나 선형적으로 연관되어 있는지 측정
- 이후 국소적 관계가 가장 잘 보존되는 훈련 세트의 저차원 표현 찾음.
- 잡음이 많지 않은 다양체를 펼치는 데 잘 작동

# 예제: 롤케이크

예제: 롤케이크

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

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



# 다른 차원 축소 기법

---

- 랜덤 사영
- 다차원 스케일링
- Isomap
- t-SNE
- 선형 판별 분석