CHAPTER

8

차원 축소

둘러보기

1. 차원의 저주
2. 차원 축소를 위한 접근 방법
3. PCA
4. 커널 PCA
5. LLE
6. 다른 차원 축소 기법

0. 기초 용어

차원의 저주(curse of dimensionality)

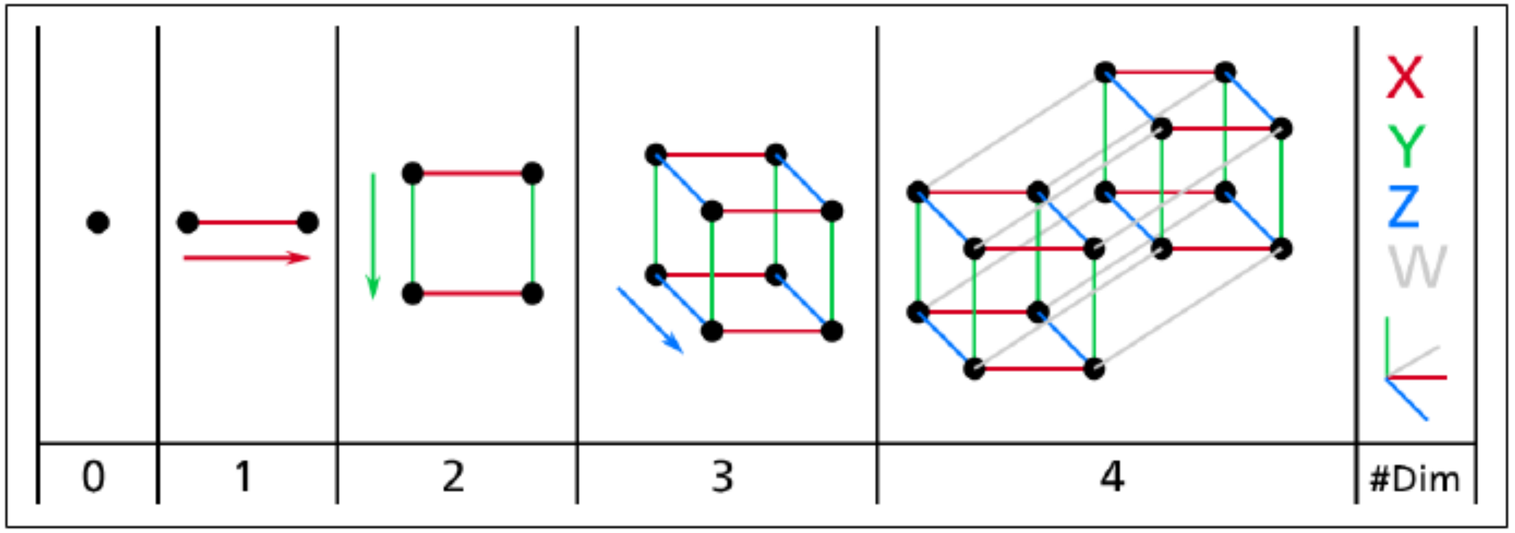
머신러닝 문제 중, 훈련 샘플이 각각이 수천, 수백만 개의 특성을 가지고 있는데 이것들이 훈련을 느리게 만들거나 좋은 솔루션을 찾기 어렵게 만드는 문제.

차원을 축소하면 일부 정보가 유실되는데, 중요하지 않은 데이터의 유실은 크게 중요하지 않으므로 이를 활용하면 훈련 속도를 높일 수 있다. 차원 축소는 데이터 시각화에도 유용하다.

1. 차원의 저주

차원

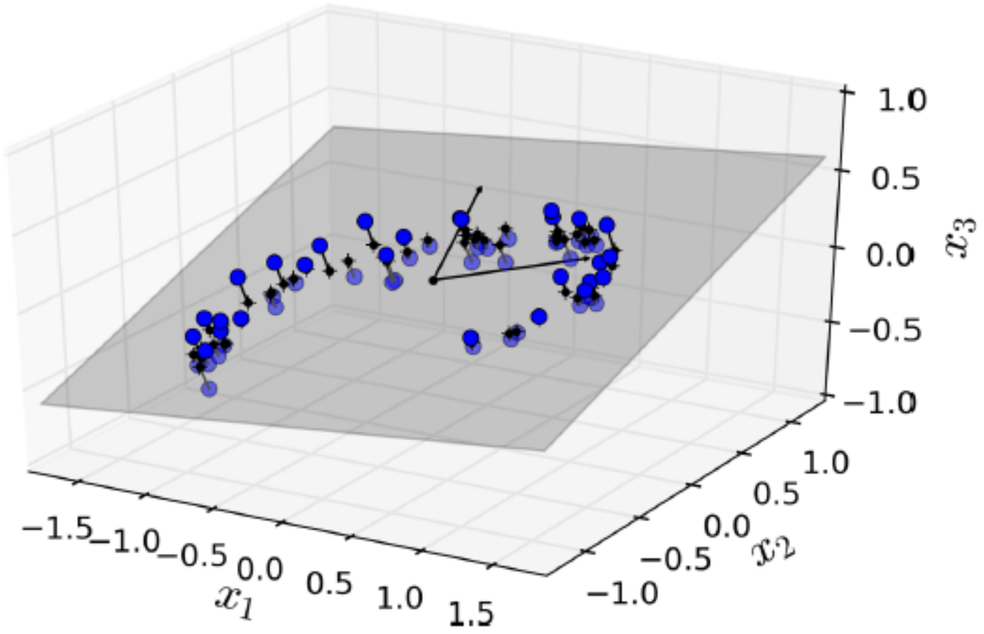
데이터셋이 고차원일수록 대다수의 훈련 데이터가 멀리 떨어져 있으며 새로운 샘플도 훈련 샘플과 멀리 떨어져 있을 가능성이 높다. 또한 훈련 세트의 차원이 클수록 과대적합의 위험도가 높아진다.

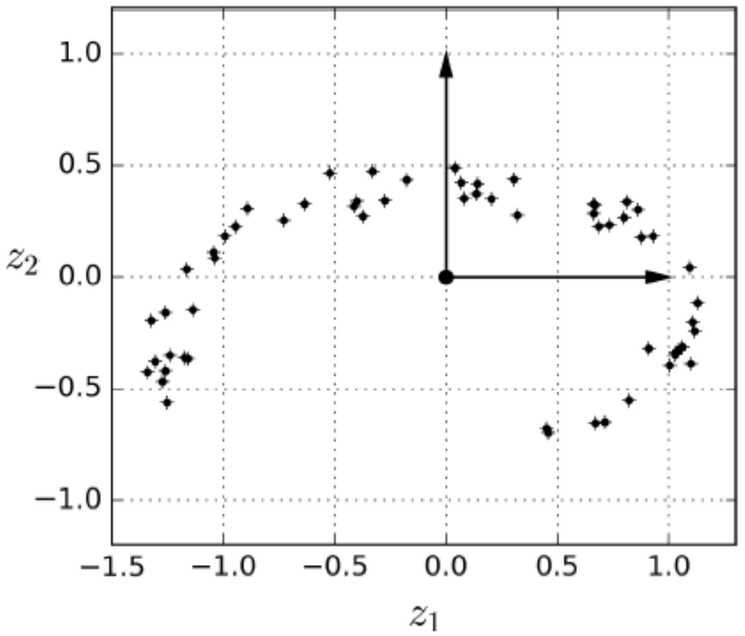


2 . 차원 축소를 위한 접근 방법

차원 축소를 위한 접근 방법 – 투영(Projection)

좌측의 경우 모든 훈련 샘플이 고차원 공간 안의 저차원 부분공간에 놓여 있음을 확인할 수 있다. 이 데이터들을 고차원 공간에서 설정한 평면를 기반으로 평면화하면 우측의 그래프와 같은 데이터셋을 얻을 수 있다. 단, 스위스 롤의 경우 투영 기법으로 축소하면 데이터가 뭉개지는 문제가 있다.





2 . 차원 축소를 위한 접근 방법

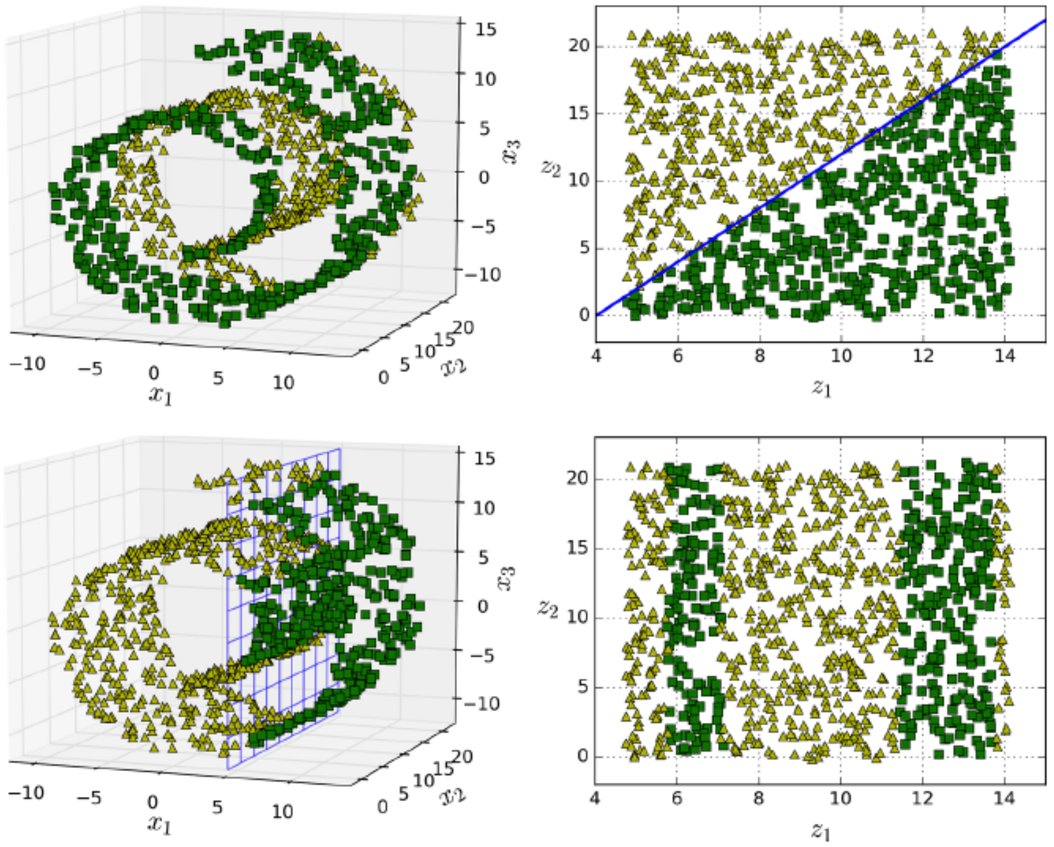
'스위스 롤'은 메니폴드의 한 예이다. 2D 메니폴드는 고차원 공간에서 휘어지거나 뒤틀린 2D 모양이다. d 차원 매니폴드는 국부적으로 d차원 초평면으로 보일 수 있는 n차원 공간의 일부이다. (d < n) 스위스 롤의 경우 d = 2, n = 3 이며 국부적으로는 2D 평면으로 보이지만 3차원으로 말려 있다.

차원 축소를 위한 접근 방법 – 매니폴드 학습

훈련 샘플에 놓여 있는 매니폴드를 모델하는 식으로 작동하는데, 이를 매니폴드 학습이라고 한다.

이는 대부분 실제 고차원 데이터셋이 더 낮은 저차원 매니폴드에 가깝게 놓여 있다는 **매니폴드 가정**에 근거한다.

매니폴드 가정



모델을 훈련시키기 전에 훈련 세트의 차원을 감소시키면 훈련 속도는 빨라지지만, 항상 더 낫거나 간단한 솔루션이 되는 것은 아니다.

3. PCA

주성분 분석 (Principal Component Analysis, PCA)

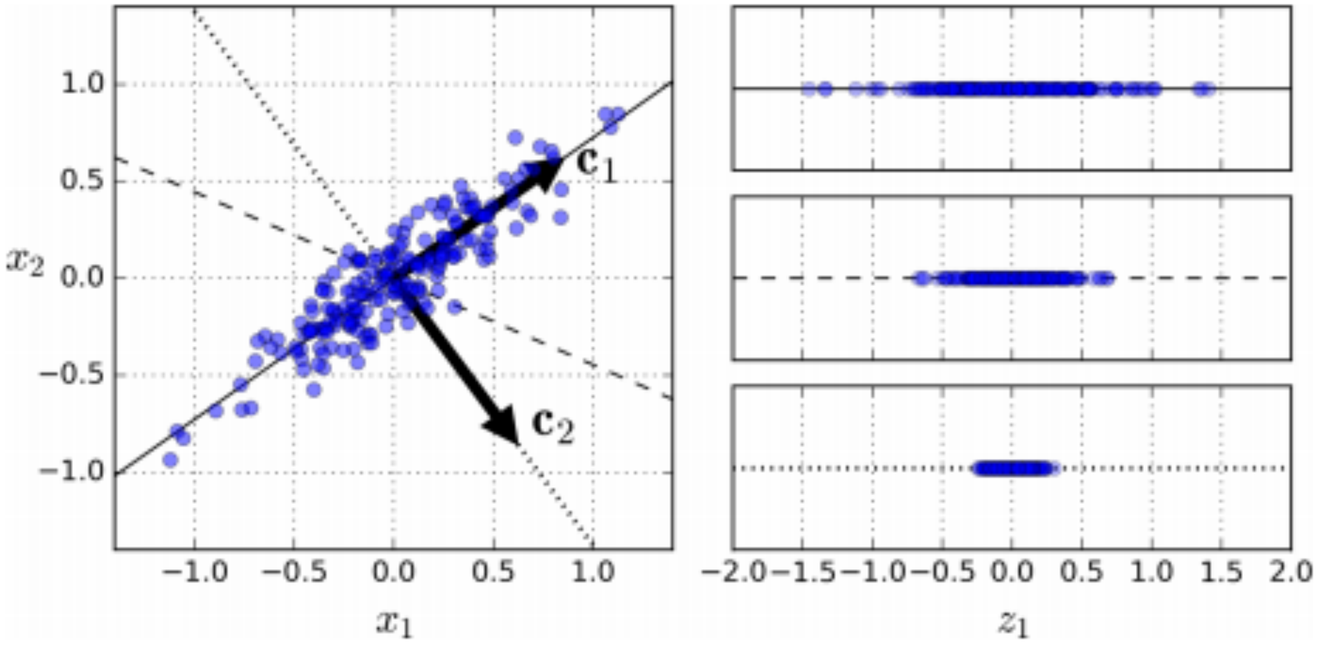
데이터에서 가장 가까운 초평면을 정의하고, 데이터를 이 평면에 투영시키는 차원 축소 알고리즘.

- 압축을 위한 PCA
- 랜덤 PCA
- 점진적 PCA
- 커널 PCA

3. PCA

분산 보존

2차원에서 1차원으로 차원을 축소하는데, 분산이 최대로 보존되는 축을 선택하는 것이 정보가 가장 적게 손실되므로 합리적이다. 이때 축은 원본 데이터셋과 투영된 것 사이의 평균 제곱 거리를 최소화하는 것이라고 설명할 수 있다.



3. PCA

주성분

이전에 언급한 '축'에 직교하고 남은 분산을 최대한 보존하는 다른 축을 찾는 것을 차원의 수만큼 반복한다. 이때, i번째 축을 이 데이터의 i번째 주성분(Principal Component, PC)라고 한다. 훈련 세트의 주성분은 특잇값 분해(Singular Value Decomposition, SVD)를 이용한다.

```
# 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]
```

▲ 파이썬 라이브러리 넘파이를 이용하여 훈련 세트의 모든 주성분을 구한 후 처음 두 개의 PC를 정의하는 두 개의 단위 벡터를 추출하는 코드.

3. PCA

d차원으로 투영하기

주성분을 모두 추출한 이후, 훈련 세트를 d차원으로 투영한다.

$$X_{d-proj} = XW_d$$

```
W2 = Vt.T[:,:2] # 2개의 주성분을 가져와
X2D = X_centered.dot(w2) # 2차원으로 투영
```

▲ 첫 두 개의 주성분으로 정의된 평면에 훈련 세트를 투영하는 코드.

3. PCA

사이킷런에서 PCA 모델을 이용하여 차원 축소

SVD 분해 방법을 사용하여 구현한다. 사이킷런은 자동으로 데이터를 중앙에 맞춰준다.

```
from sklearn.decomposition import PCA

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

pca.components_.T[:,0]
```

```
array([-0.93636116, -0.29854881, -0.18465208])
```

3. PCA

설명된 분산의 비율(Explained Variance Ratio)

explained_varience_ratio_ 변수에 저장된 이 정보는 각 주성분의 축을 따라 있는 데이터셋의 분산 비율을 나타낸다.

```
pca.explained_variance_ratio_
```

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

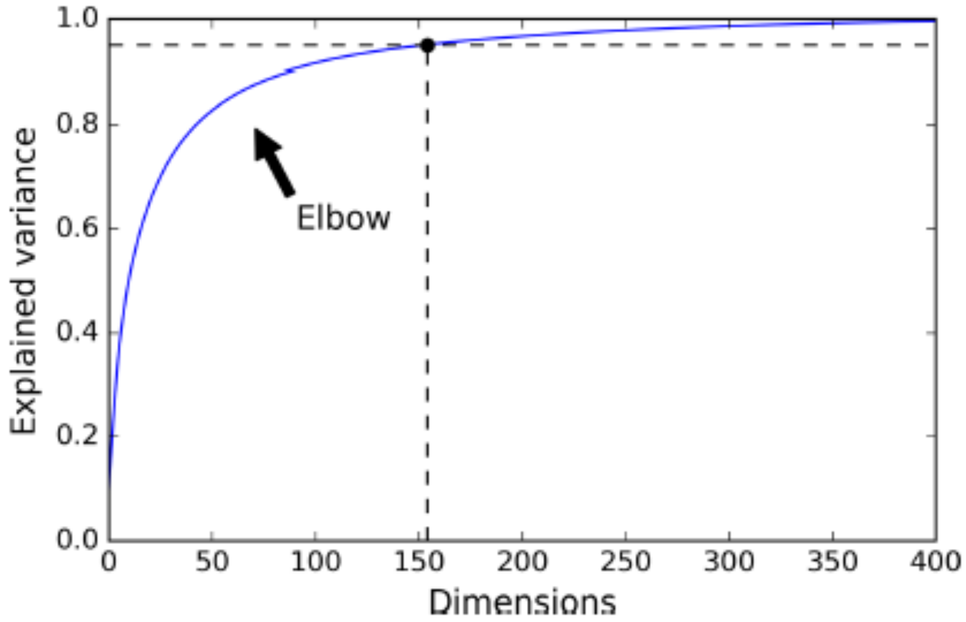
3. PCA

적절한 차원 수 선택하기

축소할 차원의 수를 임의로 정하기보다는 충분한 분산이 될 때까지 더해야 할 차원 수를 선택하는 것이 간단하다. 데이터 시각화의 경우, 차원을 2~3개로 줄이는 것이 일반적이다. 다른 방법으로는 설명된 분산을 차원 수에 대한 함수로 그리는 것이다.

```
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)
```



3. PCA

압축을 위한 PCA

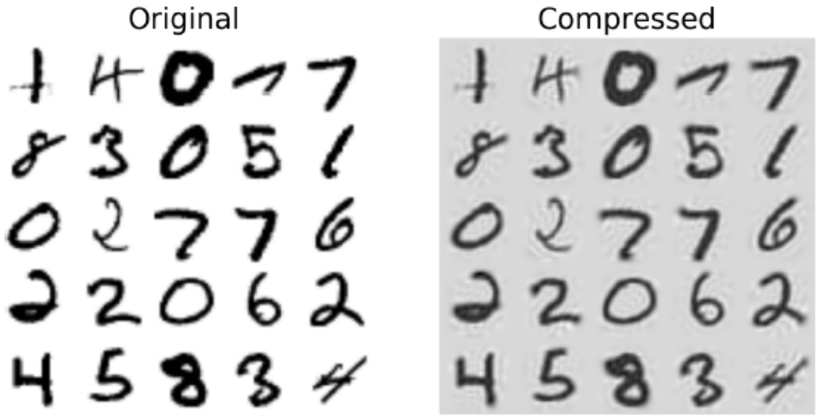
SVM과 같은 데이터셋의 크기 축소는 상당한 압축률을 보이고 분류 알고리즘의 속도를 크게 높일 수 있다. 또한 압축된 데이터셋에 PCA 투영의 변환을 반대로 적용하여 되돌릴 수도 있다. 압축으로 인해 원본을 얻을 수는 없지만 상당수 비슷한 데이터를 얻을 수 있다. 이때, 원본 데이터와 재구성된 데이터 사이의 평균 제곱 거리를 재구성 오차라고 한다.

$$\mathbf{X}_{\text{recovered}} = \mathbf{X}_{d\text{-proj}} \cdot \mathbf{W}_d^T$$

◀ 원본의 차원 수를 되돌리는 PCA 역변환공식

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

▲ MNIST 데이터셋을 154차원으로 압축하고 inverse_transform() 메서드를 사용하여 784차원으로 복원하는 코드.



3. PCA

랜덤 PCA

svd_solver 매개변수를 "randomized"로 지정하면 확률적 알고리즘을 사용해 처음 d개의 주성분에 대한 근삿값을 빠르게 찾는다. 이때 계산 복잡도는 $O(m \times d^2) + O(d^3)$ 이다. d가 n 보다 많이 작으면 완전 SVD 계산 복잡도인 $O(m \times n^2) + O(n^3)$ 보다 훨씬 빠르다.

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

3. PCA

점진적 PCA

PCA 구현의 문제는 SVD 알고리즘을 실행하기 위해 전체 훈련 세트를 메모리에 올려야 한다는 점이다. 이는 점진적 PCA 알고리즘으로 해결할 수 있다. 점진적 PCA 는 훈련 세트를 미니배치로 나눈 뒤 IPCA 알고리즘에 한 번에 하나씩 주입한다. 이는 훈련 세트가 클 때 유용하다.

```
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="")
    inc_pca.partial_fit(X_batch)

X_reduced = inc_pca.transform(X_train)
```

3. PCA

점진적 PCA

점진적 PCA 를 구현하는 또 다른 방법은 넘파이의 mammap 클래스를 사용하여 하드 디스크의 이진 파일에 저장된 매우 큰 배열을 메모리에 들어 있는 것처럼 다루는 것이다. 이 방법 역시 메모리 부족 문제를 해결할 수 있다.

```
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)
```

4. 커널 PCA

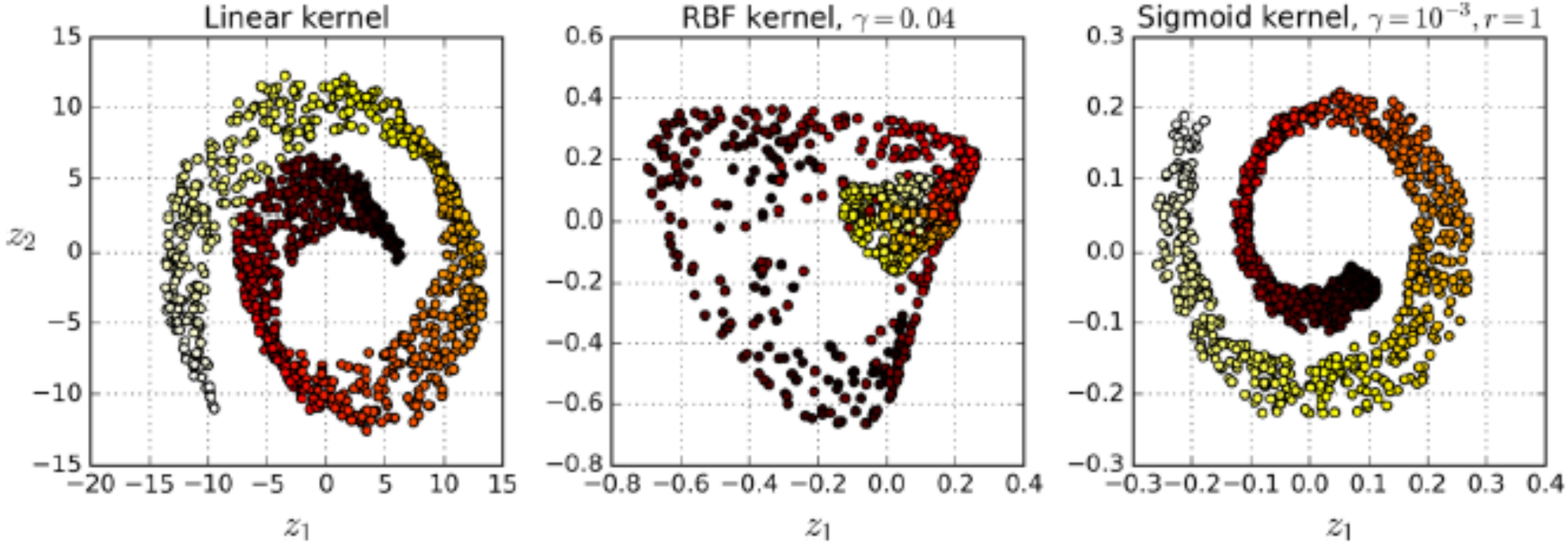
커널 트릭

샘플을 매우 높은 고차원 공간(특성 공간)으로 암묵적으로 매핑하여 서포트 벡터 머신의 비선형 분류와 회귀를 가능하게 하는 수학적 기법으로, 고차원 특성 공간에서의 선형 결정 경계는 원본 공간에서는 복잡한 비선형 결정 경계에 해당한다.

커널 PCA (kernal PCA, kPCA)

위 설명과 같은 기법을 PCA에 적용하여 차원 축소를 위한 복잡한 비선형 투형을 수행할 수 있다. 투영된 후 샘플의 군집을 유지하거나 꼬인 매니폴드에 가까운 데이터셋을 펼칠 때도 유용하다.

4. 커널 PCA



▲ (왼쪽부터) 선형 커널, RBF 커널, 시그모이드 커널을 사용하여 2차원으로 축소시킨 스위스 롤의 모습

커널 선택과 하이퍼파라미터 튜닝


```
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)
```

```
print(grid_search.best_params_)
```

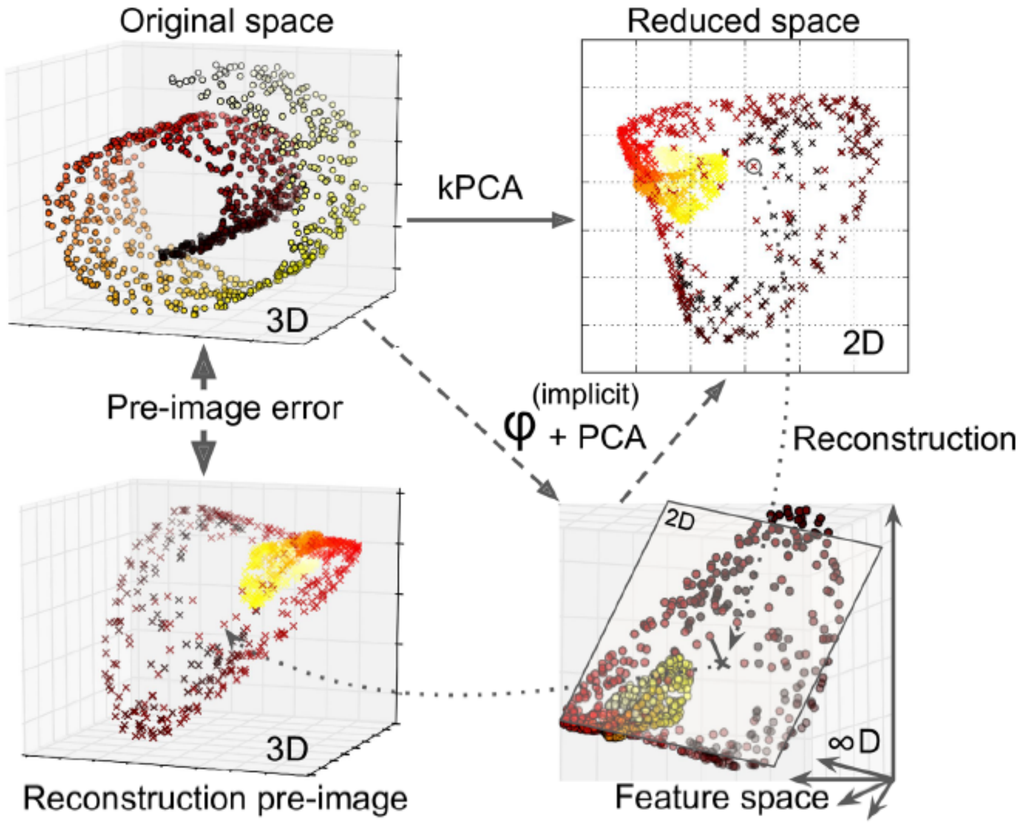
{'kpca__gamma': 0.043333333333333335, 'kpca__kernel': 'rbf'}

다음 코드는 두 단계의 파이프라인을 만드는데, 먼저 kPCA(커널 PCA)를 사용해 차원을 2차원으로 축소하고 분류를 위해 로지스틱 회귀를 적용한다. 그런 다음 파이프라인 마지막 단계에서 가장 높은 분류 정확도를 얻기 위해 GridSearchCV를 사용하여 kPCA의 가장 좋은 커널과 gamma 파라미터를 찾는다. 가장 좋은 커널과 하이퍼파리미터는 best_params_ 변수에 저장된다.

4. 커널 PCA

재구성

완전한 비지도 학습 방법으로, 가장 낮은 재구성 오차를 만드는 커널과 하이퍼파라미터를 선택하는 방식도 있다. 재구성은 선형 PCA만큼 쉽지 않다.



4. 커널 PCA

재구성 원상(pre-image)

축소된 공간에 있는 샘플에 대해 선형 PCA를 역전시키면 재구성된 데이터 포인트는 원본 공간이 아닌 특성 공간에 놓이게 된다. 이 특성 공간은 무한 차원으로 재구성된 포인트를 계산할 수 없고 실제 에러를 계산할 수 없다. 대신 재구성된 포인트에 가깝게 매핑된 원본 공간의 포인트를 찾을 수 있는데, 이를 재구성 원상이라고 한다.

4. 커널 PCA

재구성 방법

```
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)
```

▲ 투영된 샘플을 훈련 세트로, 원본 샘플을 타깃으로 하는 지도 학습 회귀 모델을 훈련시키는 코드.

```
from sklearn.metrics import mean_squared_error

mean_squared_error(X, X_preimage)
```

▲ 재구성 원상 오차를 계산하는 코드.

이렇게 되면 재구성 원상 오차를 최소화하는 커널과 하이퍼파라미터를 찾기 위해 교차 검증으로 그리드 탐색을 사용할 수 있다.

5. LLE

지역 선형 임베딩(locally linear embedding, LLE)

지역 선형 임베딩(Locally Linear Embedding)은 또 다른 강력한 비선형 차원 축소(nonlinear dimensionality reduction) 기술이다. 각 훈련 샘플이 가장 가까운 이웃에 얼마나 선형적으로 연관되어 있는지 측정하고, 국부적인 관계가 가장 잘 보존되는 학습셋의 저차원 표현을 찾는다.

```
from sklearn.manifold import LocallyLinearEmbedding

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

▲ 사이킷런의 LocallyLinearEmbedding을 사용하여 스위스 롤을 펼치는 코드.

5. LLE

LLE가 작동하는 방식

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

[조건]

$$\begin{cases} w_{i,j} = 0 & \mathbf{x}^{(j)}\text{가 } \mathbf{x}^{(i)}\text{의 최근접 이웃 } k\text{개 중 하나가 아닐 때} \\ \sum_{j=1}^{m} w_{i,j} = 1 & i = 1, 2, \cdots, m\text{일 때} \end{cases}$$

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

6. 다른 차원 축소 기법

랜덤 투영(random projection)

랜덤한 선형 투영을 사용해 데이터를 저차원 공간으로 투영한다.

- 이러한 랜덤 투영이 실제로 거리를 잘 보존한다.
- 차원 축소 품질은 샘플 수와 목표 차원수에 따라 다르며, 초기 차원수에는 의존적이지 않다.

다차원 스케일링(multidimensional scaling, MDS)

샘플 간의 거리를 보존하면서 차원을 축소한다.

Isomap

각 샘플을 가장 가까운 이웃과 연결하는 식으로 그래프를 만든다. 그런 다음 샘플 간의 지오데식 거리(geodesic distance)를 유지하면서 차원을 축소한다.

6. 다른 차원 축소 기법

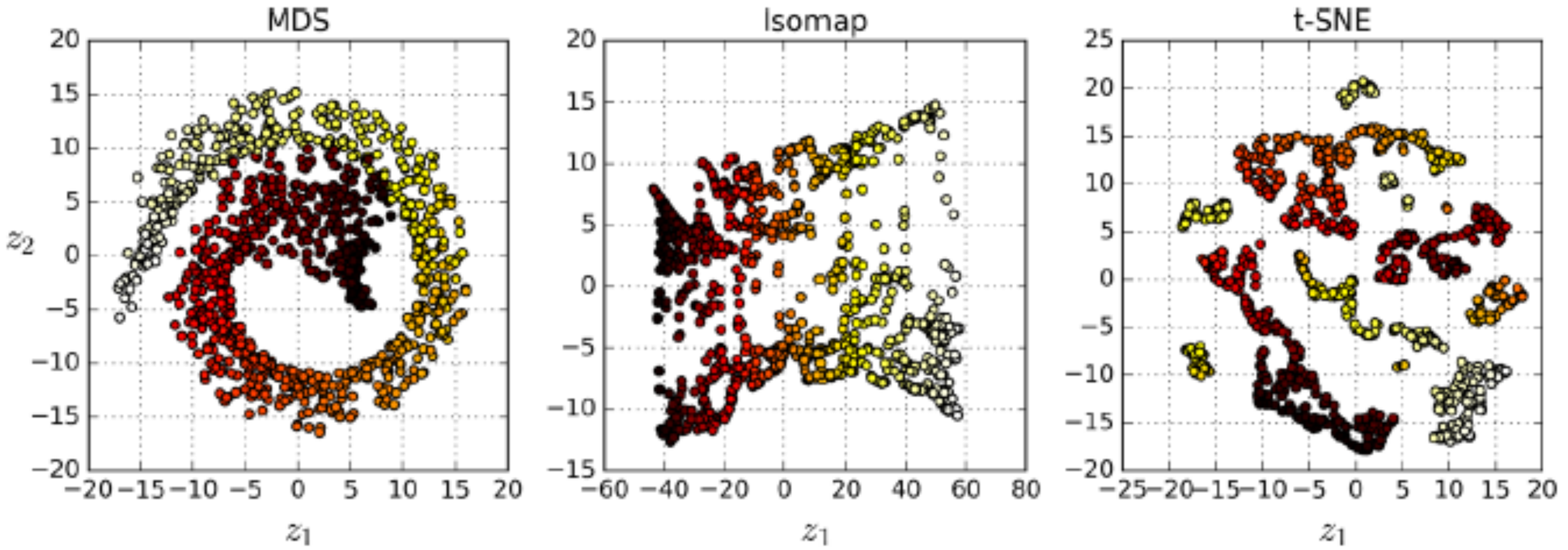
t-SNE(t-distributed stochastic neighbor embedding)

비슷한 샘플은 가까이, 비슷하지 않은 샘플은 멀리 떨어지도록 하면서 차원을 축소한다. 주로 시각화에 사용되며 특히 고차원 공간에 있는 샘플의 군집을 시각화할 때 사용된다.

선형 판별 분석(linear discriminant analysis, LDA)

분류 알고리즘이지만 훈련 과정에서 클래스 사이를 가장 잘 구분하는 축을 학습한다.

6. 다른 차원 축소 기법



▲ 여러 가지 기법을 사용해 스위스 롤을 2D로 축소하기