Chap8

Dimensionality Reduction

Written by Jin Kim

215 Intro

- 상당수의 Machine Learning 문제들은 수천~수백만가지의 Feature를 가지게 된다. 이때문에 training할때 굉장히 느리게 할 뿐만 아니라 good solution을 찾기 힘들게 된다.
- 이는 *curse of dimensionality* 라 불린다.

216 개발자 코멘트

- 차원을 축소하게 되면
  - 몇몇 데이터가 사라질 수 있다. JPEG파일을 저해상도 파일로 압축하는것과 비슷하다 생각해도 됨.
  - 트레이닝 속도가 상승한다.
  - 노이즈들이 필터될 수 있다.

216 data visualization에의 영향

- data visualization에서 굉장히 유용하다.
- 1. 차원축소로 인해 그래프가 그리기 쉬워질 수 있다.
- 2. 그 그래프로 인해 또다른 중요한 단서들을 얻을 수도 있다.
- 3. 사람들에게 실제로 데이터를 보여줘야할때도 중요하다.

216 이번 챕터 주요내용

- curse of dimensionality 에 대해 설명
- High-dimensional space는 어떻게 구성이 되어있는지
- 2가지의 dimensionality reduction 방법
  - Projection
  - Manifold Learning
- 3가지의 dimensionality reduction techniques들
  - PCA
  - Kernel PCA
  - LLE

216~217 The Curse of Dimensionality

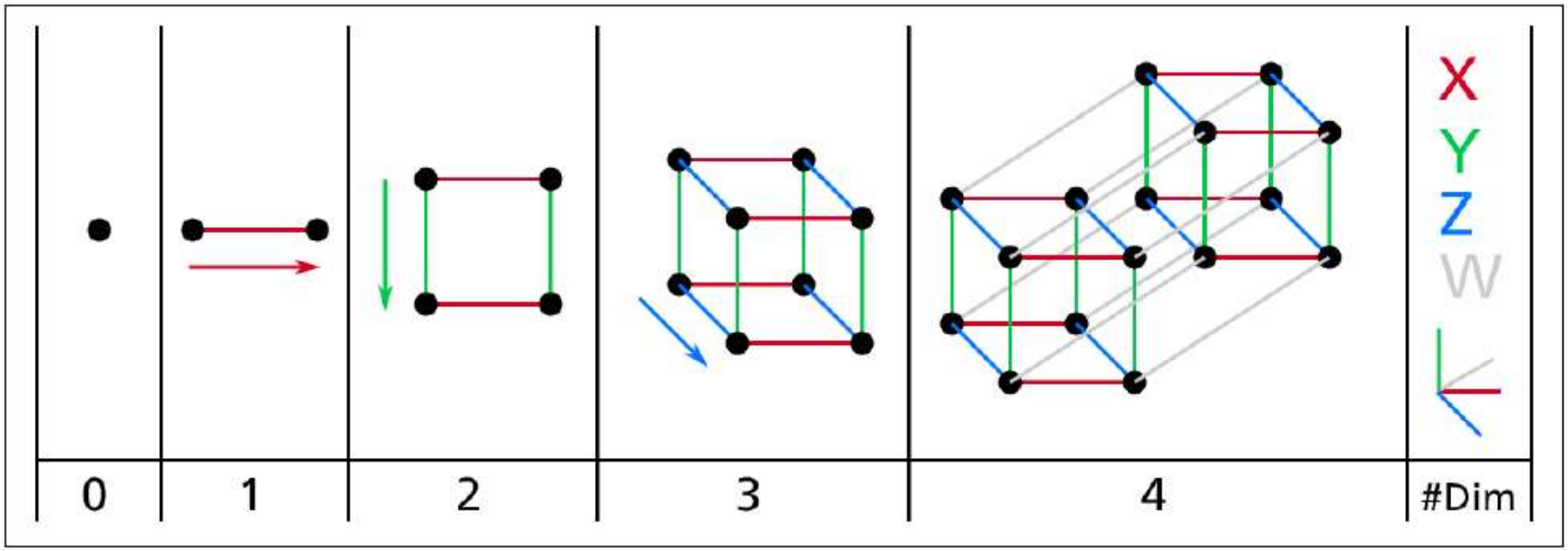


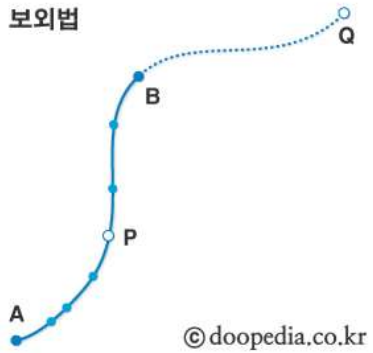
Figure 8-1. Point, segment, square, cube, and tesseract (0D to 4D hypercubes)[2]

4차원은 어떤것일지 상상하기 어렵다. 1x1 의 면에 한 점을 찍는다고 할 경우, 그 면의 테두리로부터 0.001보다 덜 떨어진 위치에 있을 확률은 0.4%정도이다. -> 1-(0.999)^2 = 0.002
그러나 10000차원에서는 확률이 99.999999%보다 더 높다. 1-(0.999)^10000 = 1-0.00004517335 = 0.99995483
고차원에서는 대부분의 지점들이 테두리에 매우 가깝다.

217 more troublesome difference

- 1x1 면의 랜덤한 두 점을 찍었을 경우 그 평균거리는 0.52정도이다.
- 3차원에서는 두 점의 평균거리는 0.66정도이다.
- 1,000,000차원에서는 408.25 (roughly 1, 000, 000/6) 정도이다.
- 즉, 선에는 가깝지만 서로간의 거리는 떨어진 상태이다.
- 이렇게 extrapolations(보외법)으로 만들어진 predictions 들은 낮은차원으로 가면 굉장히 정확도가 낮아지게 된다.

외삽법(外挿法)이라고도 한다. [그림] 에서와 같이, 곡선 위의 2점 A, B와 이 2점으로 한정된 부분 위에 몇 개의 점을 알고 있을 때, A, B로 한정된 부분 위의 다른 점 P의 위치를 추정하는 보간법(補間法)에 대하여 A, B로 한정된 밖의 부분의 점 Q의 위치를 추정하는 것을 보외법이라 한다.



217 solution?

- the training set의 사이즈를 늘린다.
  - training instances들이 충분한 density를 가지게 하기 위함.
  - 그러나 실제로는 충분한 density를 가지게 하기 위한 instance는 차원에 따라 기하급수적으로 늘어나기 때문에 힘듦.
  - 100차원에서 랜덤하게 떨어져있는 instance들이 평균 0.1만큼 서로 떨어져있게 하기 위해서는 관측가능한 모든 우주에서의 원자 수보다 더 많은 instance가 추가되어야 함.
  - -> 불가능

218~219 solution 1. projection

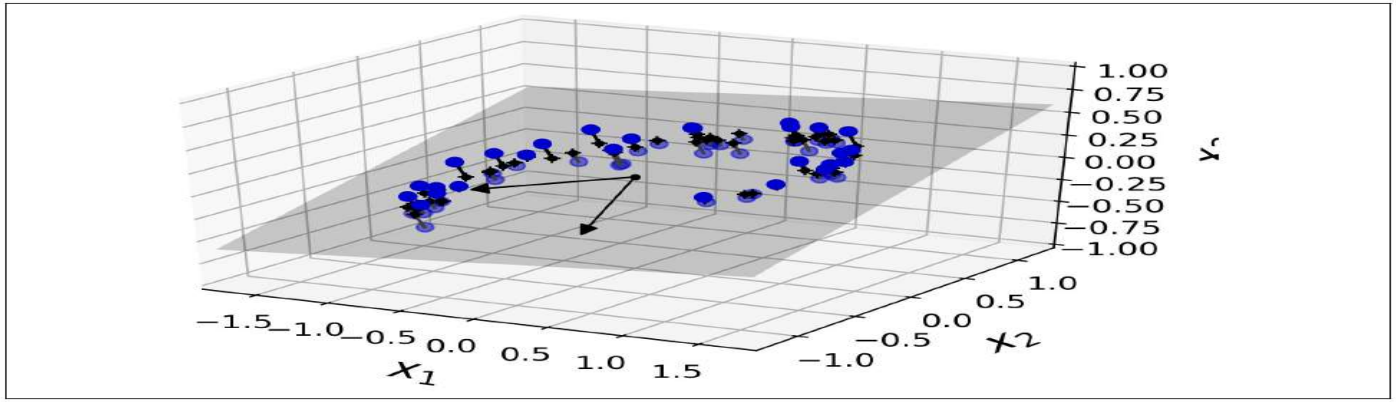


Figure 8-2. A 3D dataset lying close to a 2D subspace

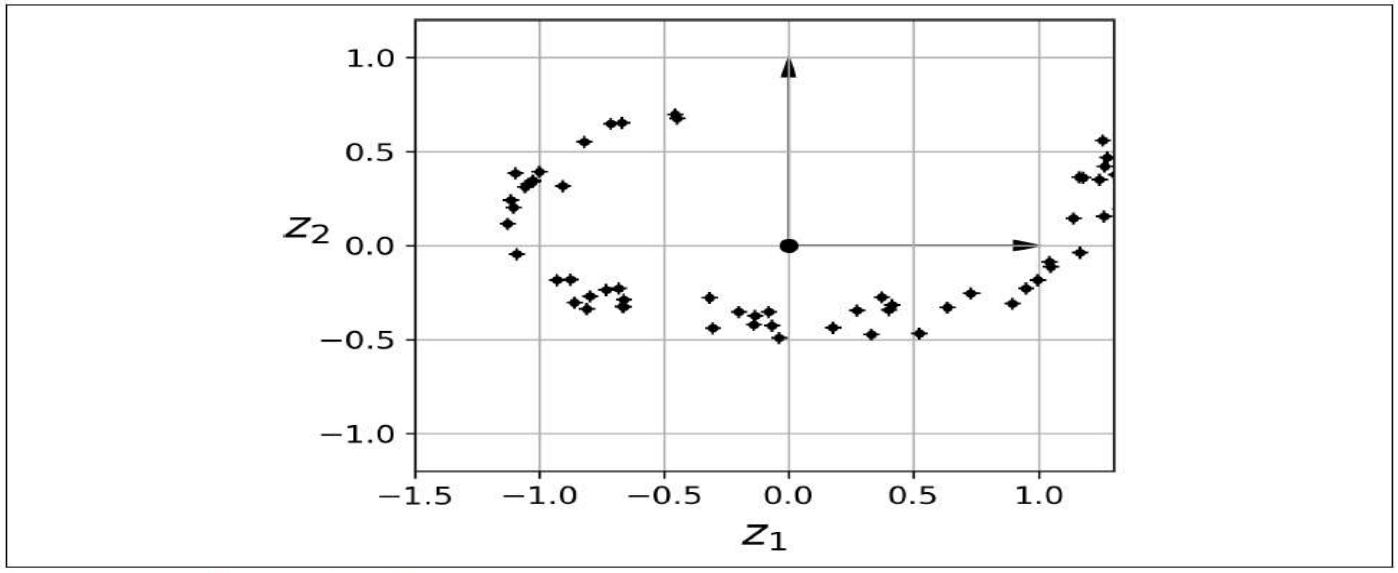


Figure 8-3. The new 2D dataset after projection

실제 세계에서는 training instance들이 랜덤하게 마구 떨어져있지는 않다.
서로 붙어있는 instance들과 관련이 있는 경우가 많다.
실제로 training instance들은 훨씬 적은차원의 부분차원(subspace)과 붙어있다.
좌측 위 그림의 경우, training instance들이 3차원 상에 있지만 2D평면상으로부터 거의 붙어있다.
Training instance들을 해당 평면에 수직으로 투영시켜 2D로 만들면 좌측 아래 그림과 같이 된다.

219~220 solution 1. projection

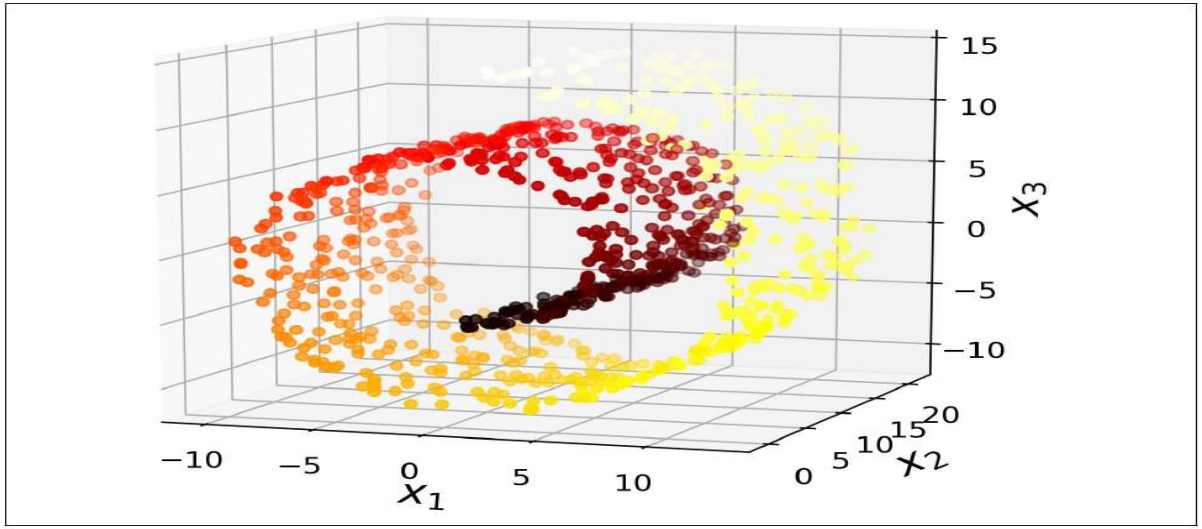


Figure 8-4. Swiss roll dataset

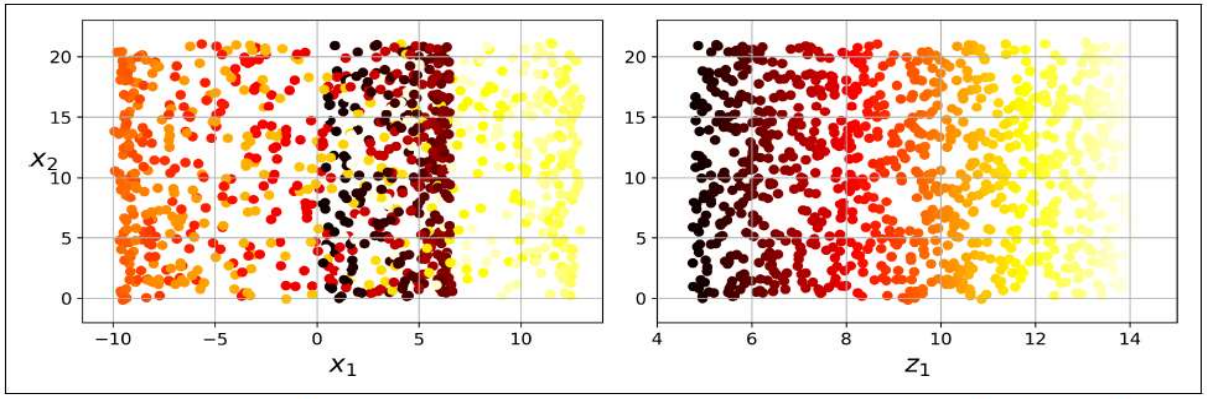


Figure 8-5. Squashing by projecting onto a plane (left) versus unrolling the Swiss roll (right)

그러나 위와 같은 방법이 항상 최선인것은 아니다. 많은 경우 좌측의 그림과 같이 뒤틀려 있는데, 같은 방법으로 2D로 투영하게 되면 데이터는 아래 좌측그림과 같이 나타나게 된다.
그러나 우리가 원하는것은 아래 우측그림과 같은 데이터이다.

-> Manifold Learning로 해결함. 다음페이지

220 solution 2. Manifold Learning

- 일반적으로 *d*-dimensional manifold 는 *n*-dimensional (*d* < *n*)의 일부이며, 이 *d*-dimensional manifold 는 *d*-dimensional hyperplane와 유사하다.
- (초평면(**Hyperplane**)은 어떤 공간 이 있을 때, 이 공간의 한 점을 통과하는 해집합을 말한다.)
- 전페이지의 그림은 $d = 2$, $n = 3$인 상태임. 2D처럼 보이지만 3D상에서 만들어짐.

220 용어설명

- *Manifold* : training instance들이 투영되는곳
- *Manifold Learning* : 차원축소 algorithms 들이 작용하는 과정
- *Manifold assumption(= manifold hypothesis)* : 고차원에서 저차원 manifold로 넘어간 실제 세계의 데이터를 가지고있는것. 경험을 바탕으로 정해지게됨

220 solution 2. Manifold Learning

- 거의 모든 Image들은 공통점이 있다.
  - 테두리가 흰색이거나, 선으로 이어져있거나, 중심부분에 그려져있음.
- 어떤 그림을 그린다고 할때, 실제로 선이 그려지는 부분은 전체 이미지에 비해 굉장히 작은 부분임.
- 디지털 이미지를 만들려고 할 때의 DOF는 실제 만들려고 하는 이미지의 DOF보다 낮을 것이다. -> 즉, 낮은 dimensional manifold를 이용하여 축소되게 됨.
- Manifold assumption은 '낮은차원에서는 regression이나 classification 이 좀 더 쉬울것이다' 라는 전제로 실행하는것이다.

221 solution 2. Manifold Learning

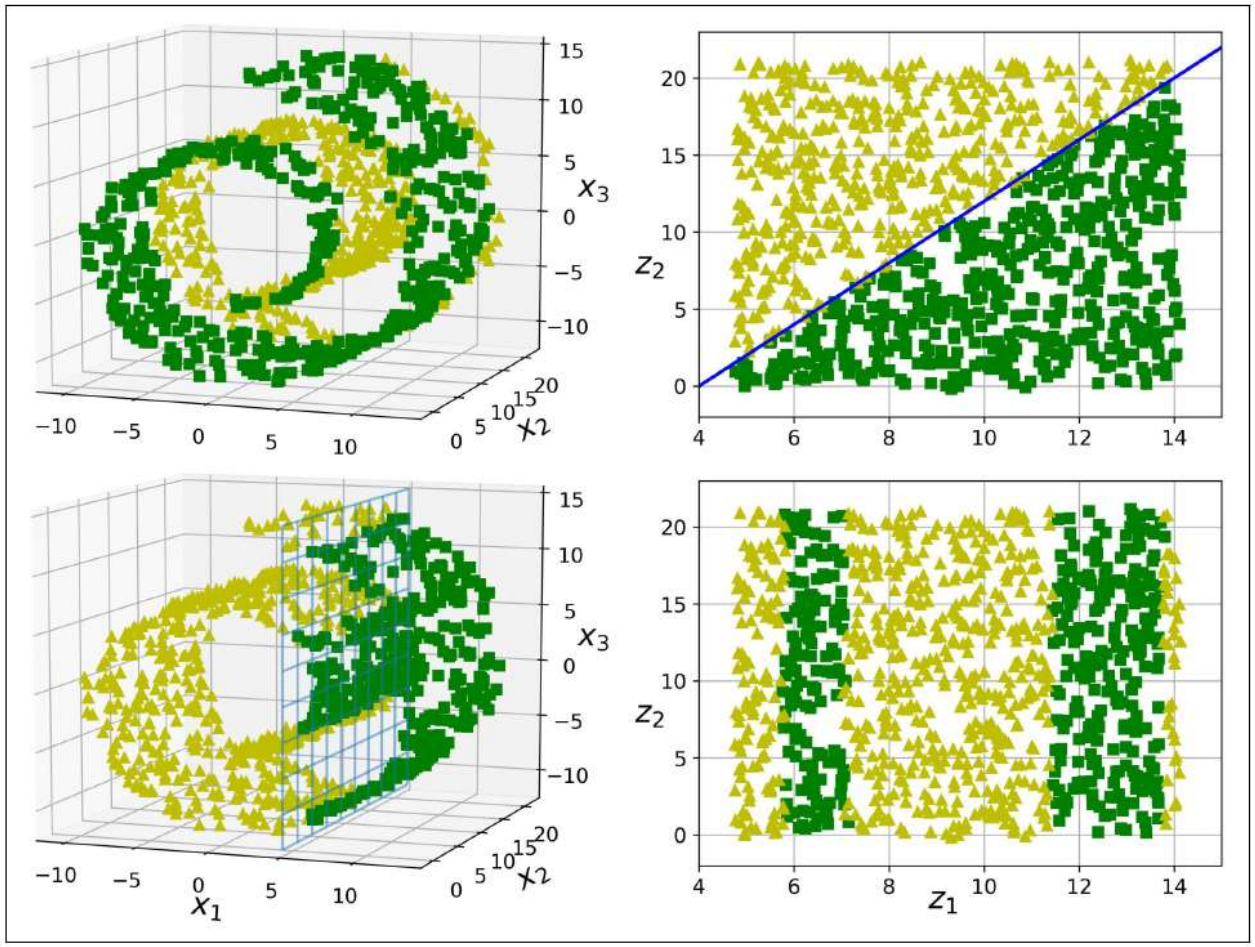


Figure 8-6. The decision boundary may not always be simpler with lower dimensions

좌상단의 그래프를 보면 decision boundary가 굉장히 복잡해 보인다.
그러나 우상단처럼 만들게되면 쉽게 알 수 있음.

근데 반대로, 좌하단과 같은 경우에는 x1=5일때 단순하게 3D상에서는 나눠 질 수 있지만, 우하단처럼 manifold를 이용해 나누게되면 더 복잡해지게 된다.

즉, 항상 차원축소가 좋은것만은 아니란뜻.

222 *Principal Component Analysis* (PCA)

- 현존하는 가장 유명한 차원축소 방법.
- 위에서 배운 방법과 같이 hyperplane(초평면)에 데이터를 projection 해서 축소시키는 방법임.

222 Preserving the Variance

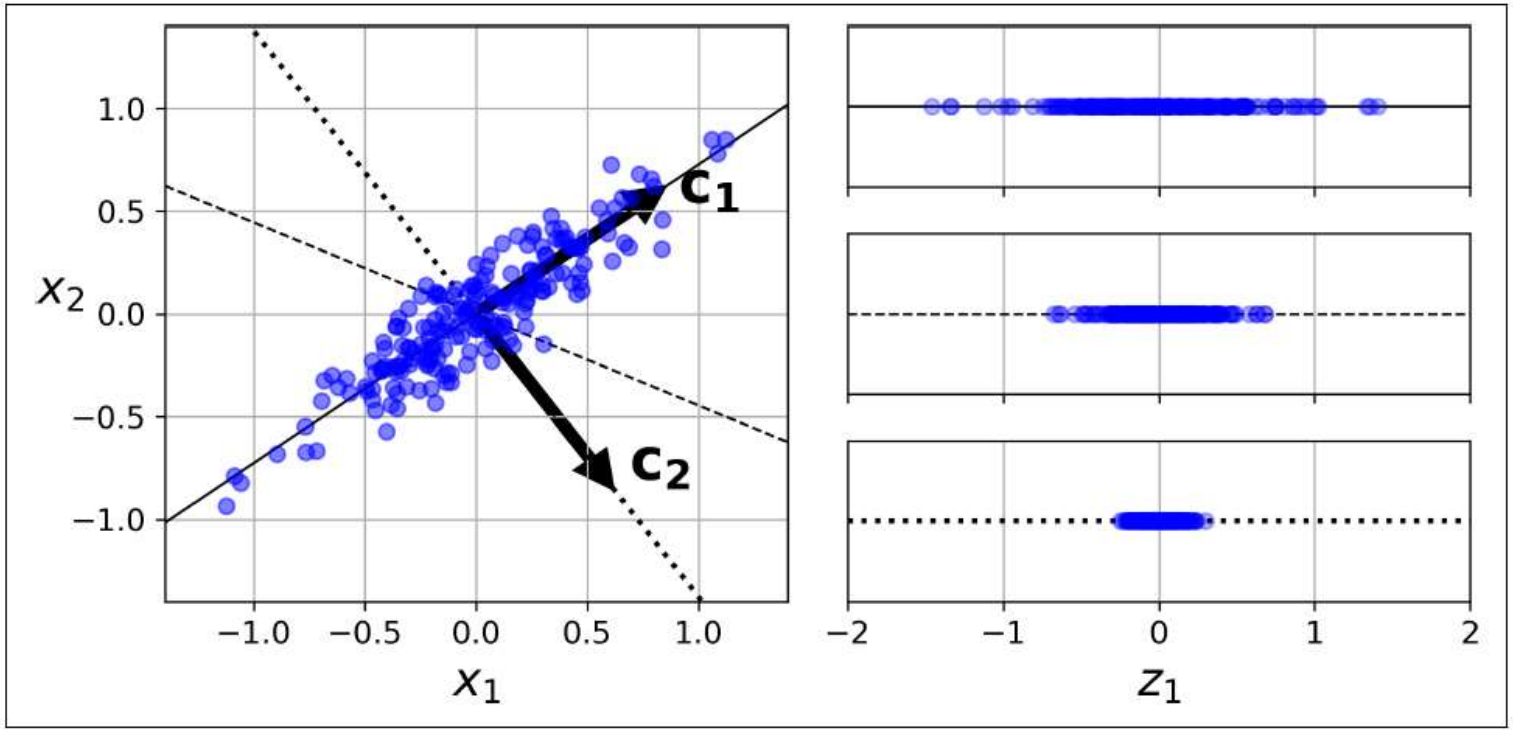


Figure 8-7. Selecting the subspace onto which to project

맨 먼저 어떤식으로 hyperplane을 적용할건지 생각해봐야함.

C1선을 기준으로 projection을 하게되면 최대의 variance를 가지게 된다.
C2선이 기준이라면 매우 적은 varianc임.

가장 variance가 높은 축을 선택하는게 좋아보인다.
쉽게 할 수 있는 방법중 하나는 좌측과 우측의 mean squared distanc를 최소로 만드는것임.

223 Principal Components(PC)

- PCA는 가장 큰 variance를 가질 수 있도록하는 축을 찾아낸다.
- 그 후에, 해당 축에 대해 수직인 또 다른 축을 찾음.
- 근데 앞전 페이지 그림은 2D였기 때문에 축을 두개만 찾고 더이상 찾지 않음.
- 고차원일수록 더 많은 축을 찾아냄.
  - Ex. 3차원이다 -> axis1을 찾음. 그거에 대해 수직인 axis2와 모두에 대해 수직인 axis3을 찾음.만약 4차원이면 위 모두에 대해 수직인 axis4도 찾았을것.

223 Principal Components(PC)

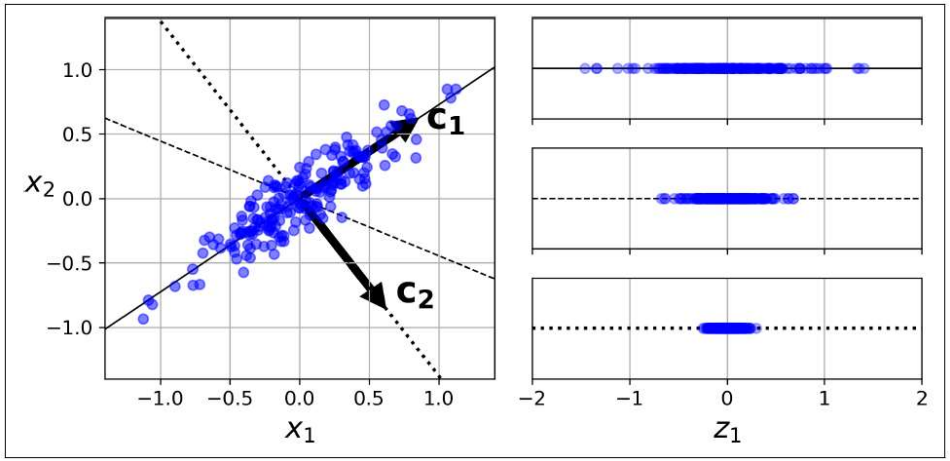


Figure 8-7. Selecting the subspace onto which to project

ith axis를 ith *principal component* (PC).라고 부름.
좌측 그림(8-7)에서 1st PC는 C1, 2nd PC는 C2임.

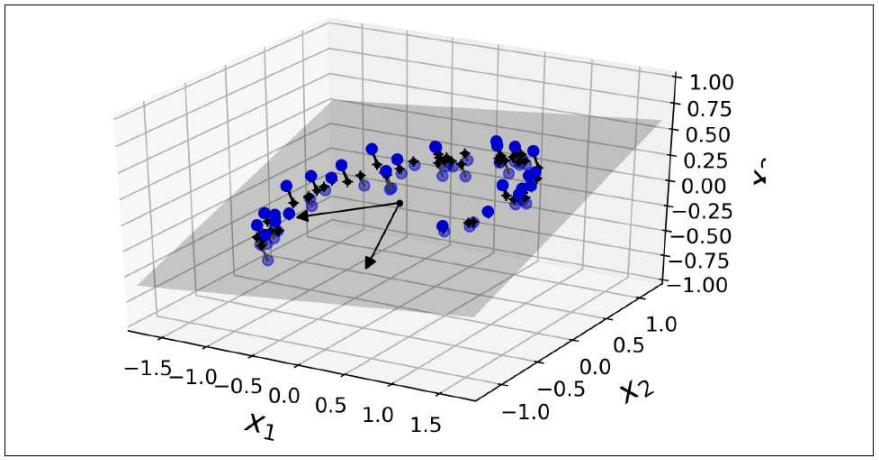


Figure 8-2. A 3D dataset lying close to a 2D subspace

ith axis를 ith *principal component* (PC).라고 부름.
좌측 그림(8-2)에서 1st 와 2nd PC는 평면에, 3rd PC는 평면에 수직으로 생길것임.

223 개발자 코멘트

- PC의 방향은 안정적이지 않음.
- 데이터를 살짝 바꾼 후 PCA를 다시 실행하면, 완전히 반대방향을 가리키는 PC가 생성될 수도 있음.
- 그러나, 일반적으로 해당 평면에 투영되는 데이터는 같음.

223 *Singular Value Decomposition* (SVD)

Equation 8-1. Principal components matrix

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

SVD를 이용하여 training set matrix $\mathbf{X}$ 를 3개 matrix 의 곱으로 나타낼 수 있음. $\mathbf{U}\,\Sigma\,\mathbf{V}^T$

V는 우리가 필요한 모든 PC를 가지고 있음.

아래는 python code임

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

224 Projecting Down to *d* Dimensions

- d개의 PC를 정했으면, d차원으로 투영하여 차원을 축소 할 수 있다.
- 최대한 variance를 보존하는 방향으로 PC를 정한다.
  - 원래 형태처럼 유지하는것이 좋음.

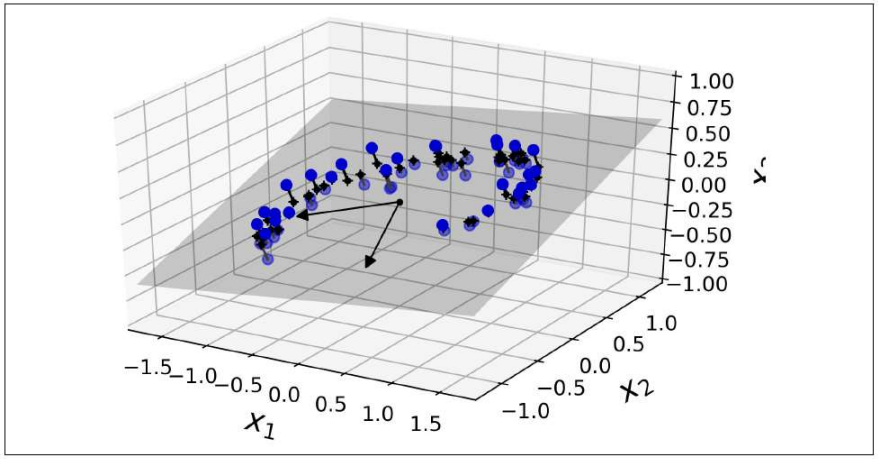


Figure 8-2. A 3D dataset lying close to a 2D subspace

224 matrix multiplication(행렬곱) 구하는 방법

Equation 8-2. Projecting the training set down to d dimensions

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

위에서 말한대로 d개의 PC를 구하기 위하여
Training set matrix X에 d개의 column 으로 구성된 Wd 행렬을 곱하면 됨.
아래는 python code

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

224~225 Using Scikit-Learn

```
from sklearn.decomposition import PCA

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

위와 같은 코드로 PCA를 python에서 구현 가능
그 후에
pca.components_.T[:,0]
와 같은 코드로 첫 번째 PC를 얻을 수 있음.

225 Explained Variance Ratio

- 각 PC의 *explained variance ratio* 또한 매우 중요한 정보를 가지고 있음.
- -> data variance가 해당 축을 따라 투영되는 비율을 나타냄.

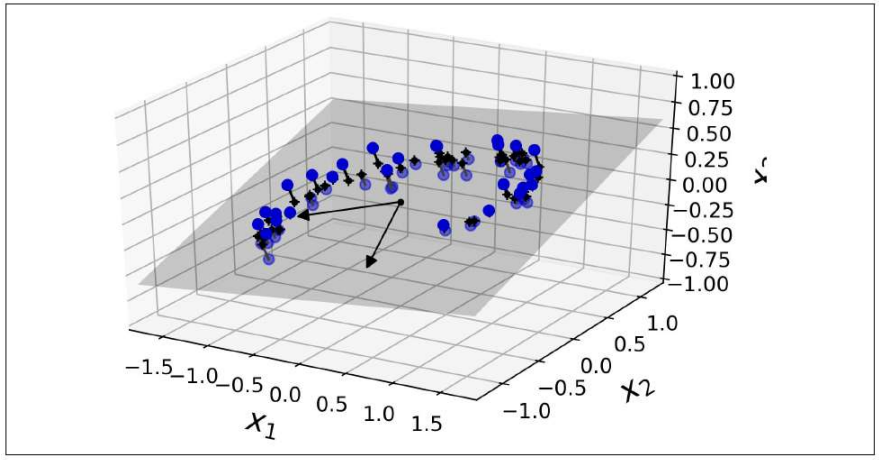


Figure 8-2. A 3D dataset lying close to a 2D subspace

좌측 그림과 같은 경우, explained variance ratio는 아래와 같이 나타남

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

그 뜻은, 84.2%의 dataset's variance 가 첫 번째 axis를 통해 투영됐고, 14.6%가 두 번째 axis를 통해 투영됐다는 뜻임. 1.2%정도만이 세 번째 axis를 통해 투영됨. 1.2%정도면 적은 information을 가지고 있으므로 차원축소가 괜찮게 됐다고 볼 수 있음.

225 Choosing the Right Number of Dimensions

- 차원의 수를 마음대로 정하기 보다는, 위의 파라미터를 참고하여 95%이상의 variance가 사용 될 수 있게 정하는것이 좋다.
  - Visualization인 경우는 예외. 이 경우는 2~3차원으로 줄여야 하기때문.

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

- 위 코드는 95%이상의 variance를 유지하면서 가장 최소한의 차원으로 줄이는것임.
- 그 후에 n_components=d 로 설정하고 PCA를 다시 돌리면 됨.

위와같이 n_components를 수동으로 설정하는 방법도 있지만, variance ratio로 설정하는 방법도 있음.

225 Choosing the Right Number of Dimensions(variance ratio설정)

- n_components에 차원의 수를 입력하는 대신, 0~1사이의 수를 입력하여 variance ratio로 설정 할 수도 있다.
  - 예를들어 0.95를 입력하면 variance가 95% 될때까지의 최소 차원으로 줄임.

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

225~226 Choosing the Right Number of Dimensions(plot그려서)

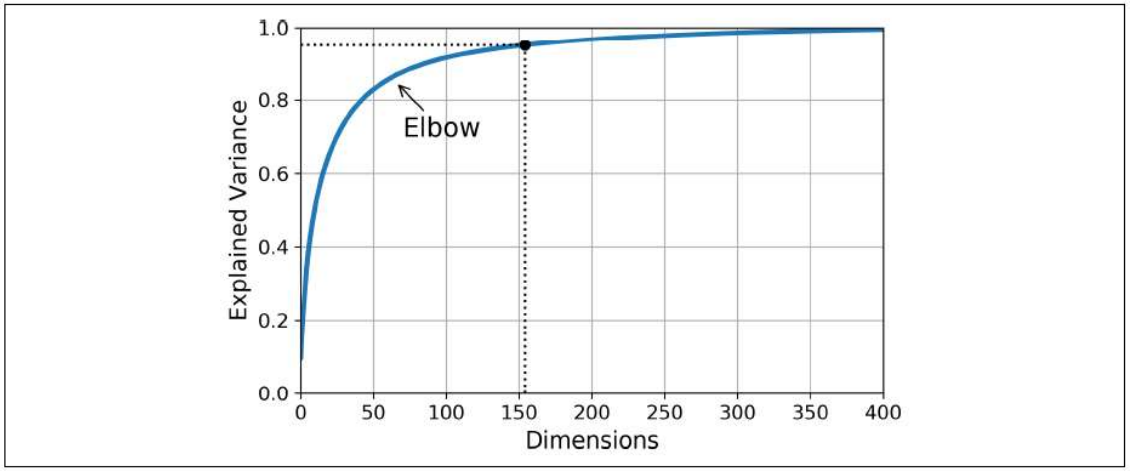


Figure 8-8. Explained variance as a function of the number of dimensions

좌측과 같이 plot을 그리는 방법도 있다.
Cumsum을 plot으로 그려보면 좌측과 같은 그림으로 나온다.
보통은 기울기가 급격히 변하는 elbow부분이 있는데, 이 부분을 지나게 되면 variance증가량이 줄어든다.

100차원으로 줄여도 variance는 준수한것으로 나타난다.

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

226 PCA for Compression

- 전 페이지의 그래프를 보면, 기존 784개의 feature를 150개 가량으로 줄였다.
- inverse transformation를 이용하여 줄어든 feature를 다시 원래대로 복구하는것도 가능하다.
  - 100%의 데이터를 복구하지는 못한다. Variance가 줄어든 만큼 줄어들게됨.
  - 다시 복구한 후의 데이터<->원본데이터간의 차이를 *reconstruction error* 라고 한다.

226~227 *reconstruction error*

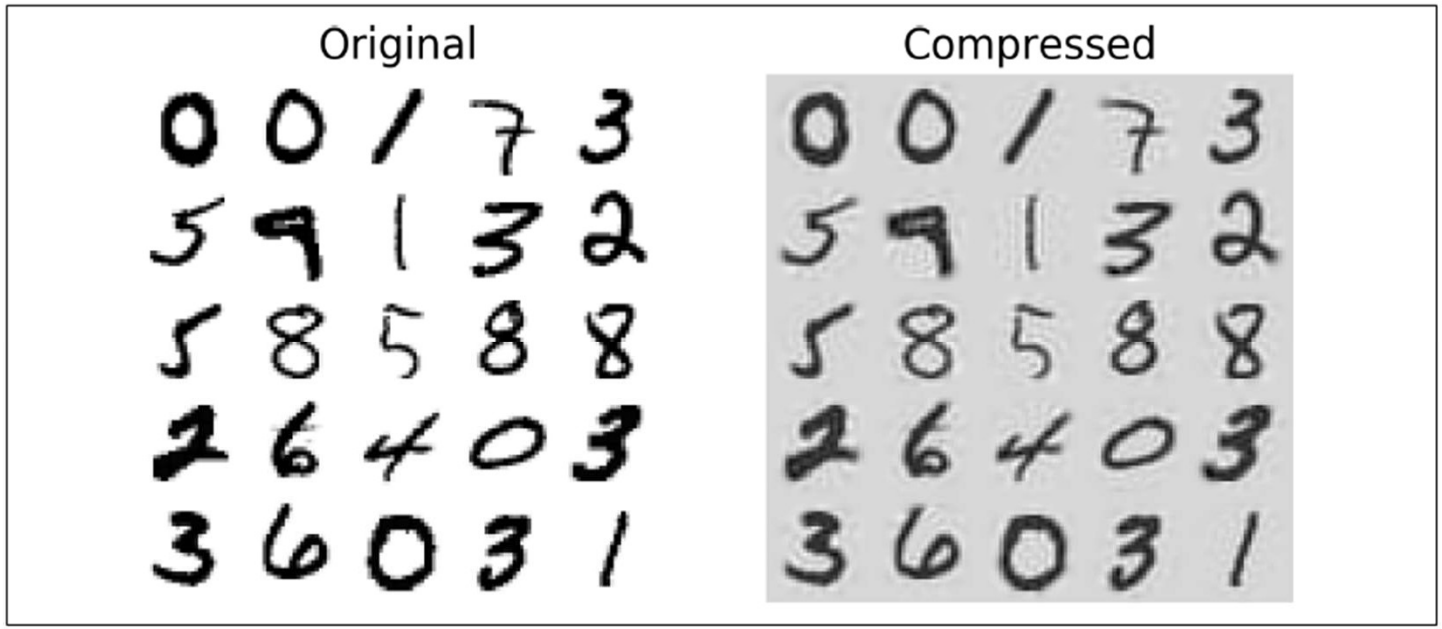


Figure 8-9. MNIST compression preserving 95% of the variance

좌측 : 원본 데이터
우측 : 784->154->784로 compression 한 후 다시decompression 하여 얻은 데이터

Equation 8-3. PCA inverse transformation, back to the original number of dimensions

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

좌측이 PCA inverse transformation의 공식

227 Randomized PCA

- svd_solver를 "randomized"로 설정했다면 사이킷런에서 stochastic algorithm을 적용한다.
  - 이는 *Randomized PCA* 라고 불림.
- 최소 차원 d를 훨씬 빠르게 찾을 수 있음.

227~228 Incremental PCA

- PCA의 문제점은 모든 training set이 메모리 위에 올라가 있어야 작동을 한다는것이었음.
- *Incremental PCA* (IPCA)는 training set을 여러개의 mini-batch로 나누어 한번에 한개의 mini batch를 차원축소 할 수 있음.

```
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)
```

좌측의 코드는 100개의 미니배치로 만들어 PCA를 적용하는것

228~229 Kernel PCA

- 5장에서 배운 nonlinear classification / regressio의 kernel trick을 이용한것임.
  - Support Vector Machines를 이용해 데이터를 고차원으로 만들어 중간을 분리시키는것이었음.
  - 고차원의 linear decision boundary 는 저차원의 nonlinear decision boundar와 같게되므로, 고차원에서 나눴었음.
- 같은 트릭을 이용하여 PCA에 적용한다. = kPCA라 부름.
- 다음페이지 계속

228~229 Kernel PCA

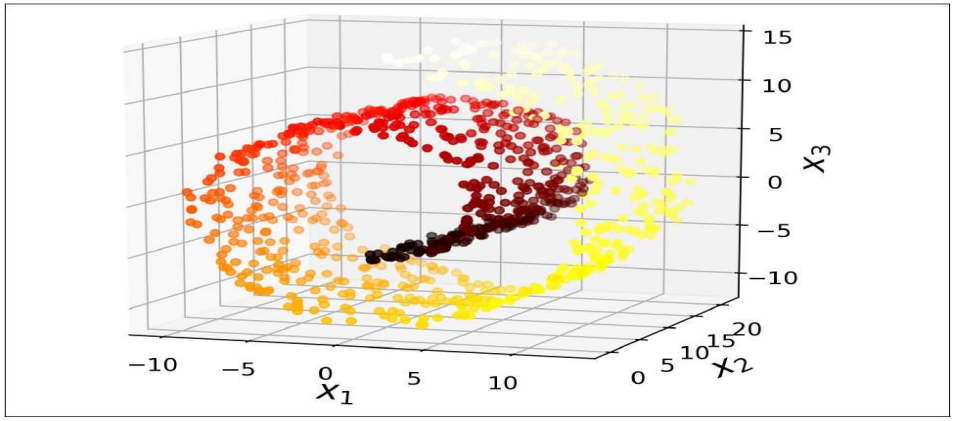


Figure 8-4. Swiss roll dataset

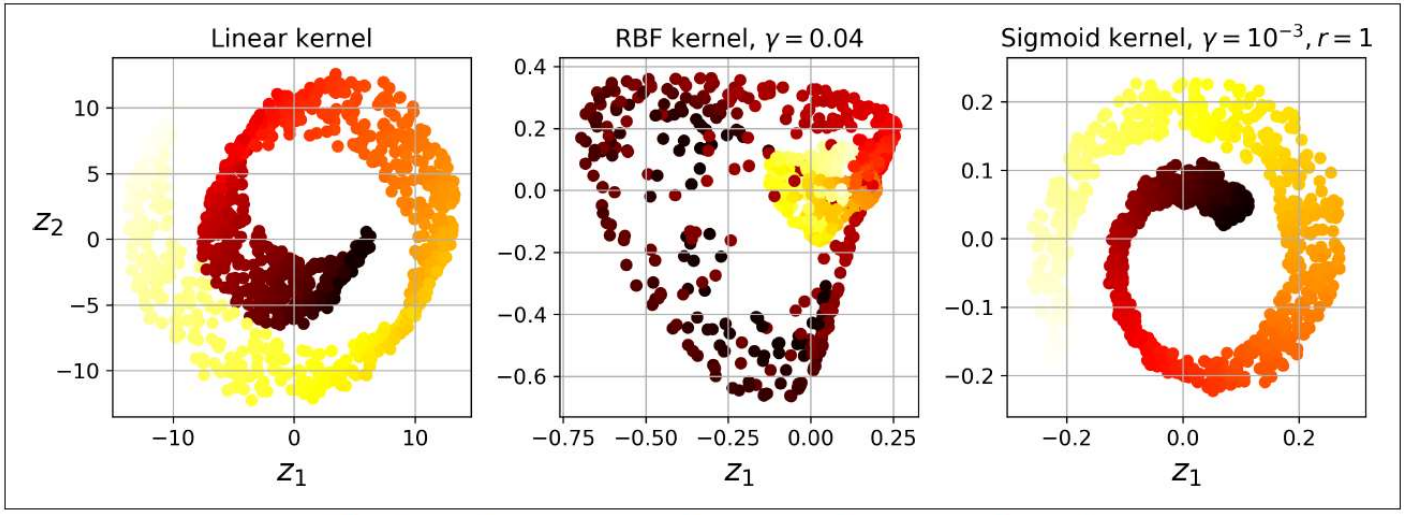


Figure 8-10. Swiss roll reduced to 2D using kPCA with various kernels

좌상단의 Swiss roll 데이터셋을

1. Linear kernel
2. RBF kernel
3. Sigmoid kernel

로 차원축소를 한 그림임.

229 Selecting a Kernel and Tuning Hyperparameters

- kPCA는 unsupervised learning algorithm임.
  - 즉, 어떤것이 best kernel인지 측정할만한것이 없음.
  - 그러나 차원축소 자체가 supervised learning task의 준비과정이기 때문에, grid search를 이용하여 해당 task가 가장 높은 성능을 가지는 hyperparameter값을찾을 수는 있음.
  - Hyperparameter 값 찾는 과정은 다음페이지

229~230 kPCA hyperparameter 설정

```
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)
```

1. 파이프라인 설정
    1. kPCA적용
    2. Logistic Regression 적용
2. GridSearchCV 사용하여 가장 좋은 kernel종류와 gamma값을 얻을 수 있음.

아래는 결과값

```
>>> print(grid_search.best_params_)
{'kpca__gamma': 0.043333333333333335, 'kpca__kernel': 'rbf'}
```

230~231 kPCA hyperparameter 설정

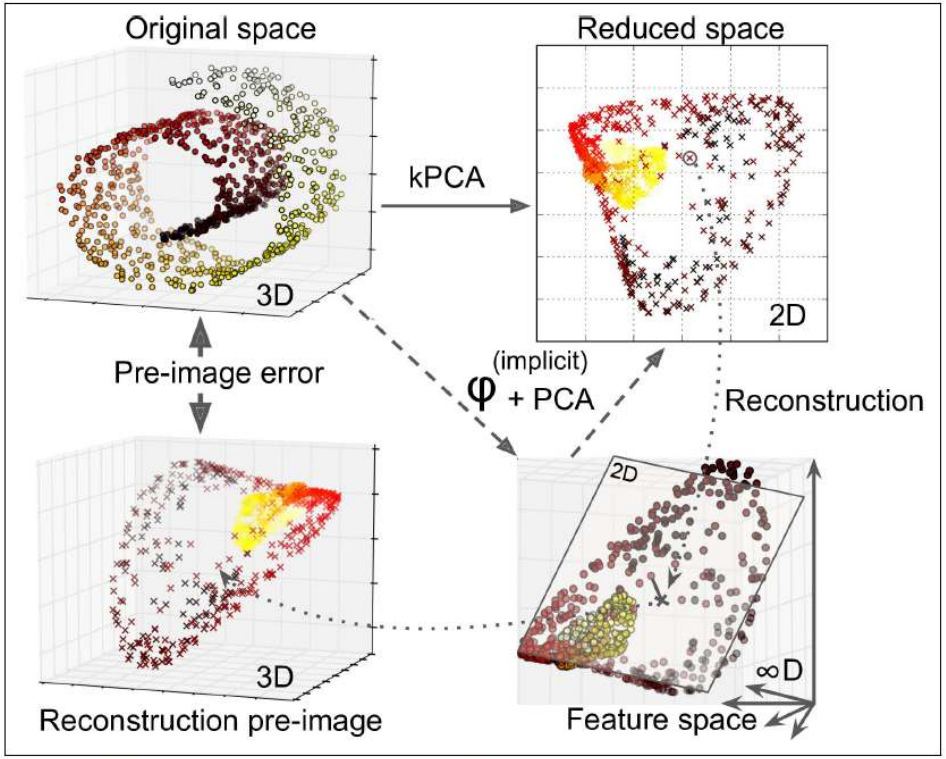


Figure 8-11. Kernel PCA and the reconstruction pre-image error

위 방법처럼 하거나, 혹은 reconstruction error 가 가장 적어지는 gamma / kernel 종류를 선택하면 되는데, linear PCA에서는 어려움이 있음.

좌상단 : 원본데이터

232~233 LLE

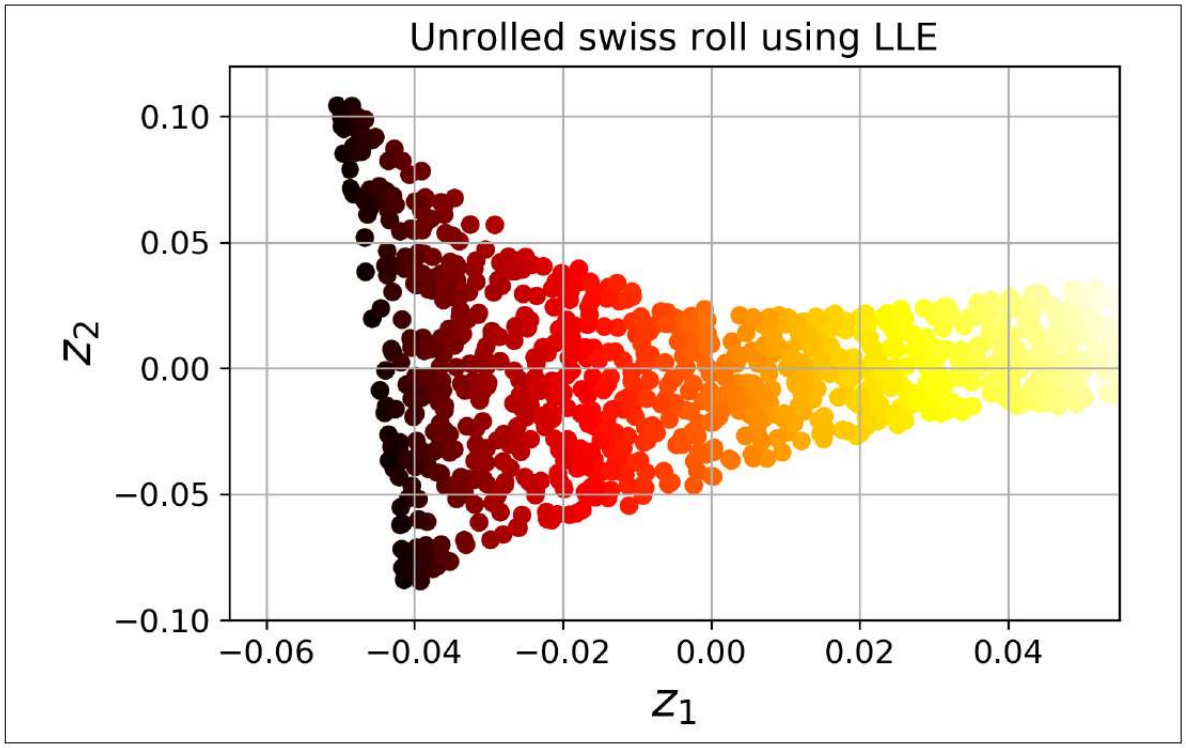


Figure 8-12. Unrolled Swiss roll using LLE

ff

234~235 Other Dimensionality Reduction Techniques

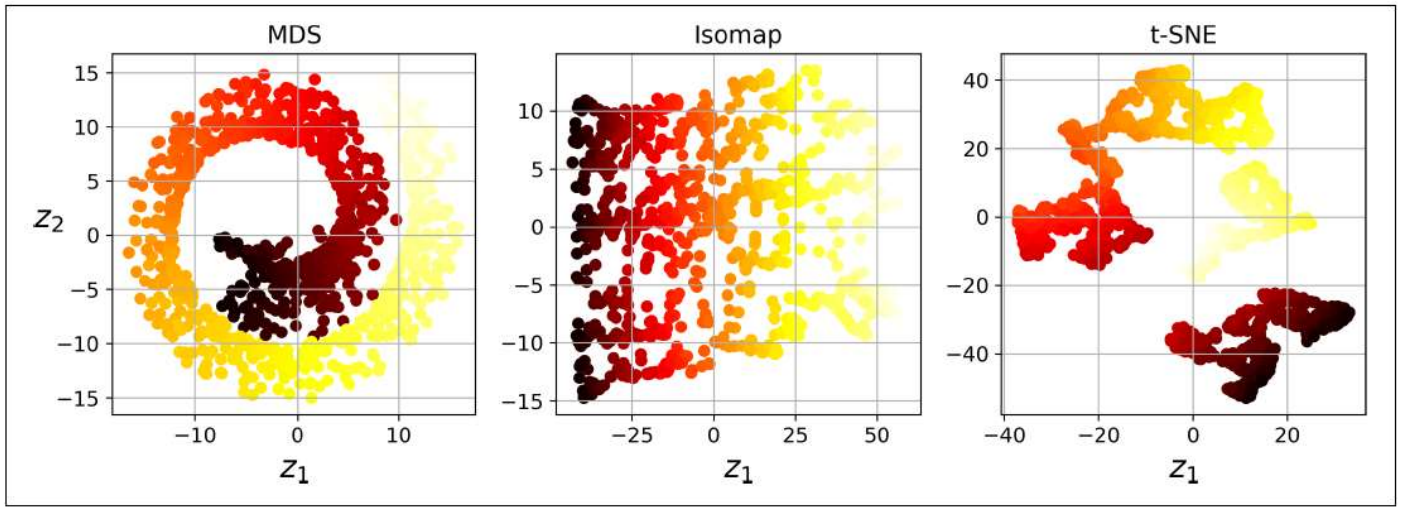
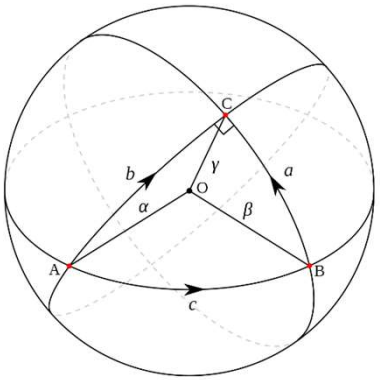


Figure 8-13. Reducing the Swiss roll to 2D using various techniques

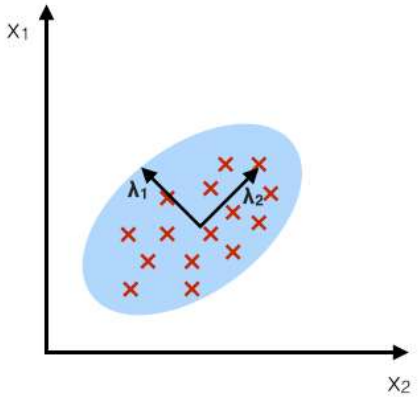
3. *t-Distributed Stochastic Neighbor Embedding* (t-SNE)
   -> 비슷한 instance의 거리는 유지, 다른 instance의 거리는 멀어지게 하는 방법으로 차원축소함. Visualization에 거의 사용됨.
   (고차원에서의 instance 군집을 나타낼때 등)

1. *Multidimensional Scaling* (MDS)
   -> Instance간의 거리를 유지하며 차원 축소
2. *Isomap*
   -> 각 instance를 가장 가까운 instance와 연결하여 그래프를 그린 후, 측지선(아래 그림 참고)의 거리를 유지하는 방법으로 차원축소

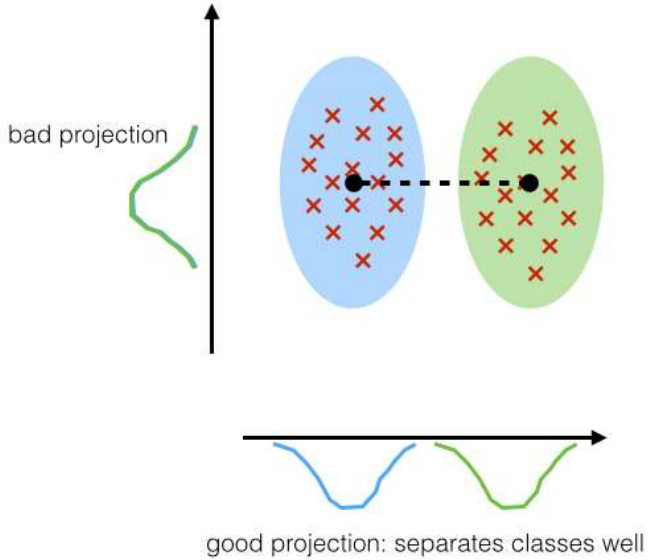


234~235 Other Dimensionality Reduction Techniques

PCA:
component axes that maximize the variance



LDA:
maximizing the component axes for class-separation



4. Linear Discriminant Analysis (LDA)
-> Classification algorithm 이며, 트레이닝 중 각 class별로 어떤 축이 가장 다른지 판별 할 수 있고, 이 다른 축을 이용해 hyperplane을 만들어 낼 수 있음.
장점은 투영시 각 클래스별로 최대한 멀리 떨어지게 할 수 있음.
SVM classifier등의 classification algorithm 을 적용하기 전에 사용하면 좋음.