Redukcja wymiarowości

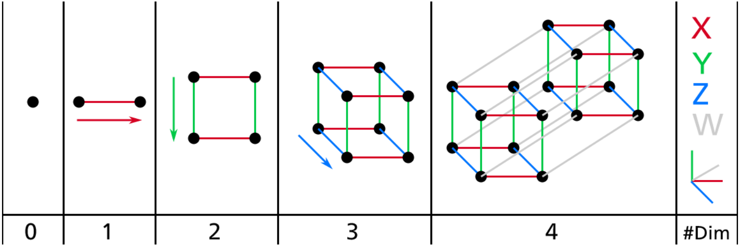
Igor Wojnicki

April 28, 2024

Plan prezentacji

Problem wymiarów

przestrzeń wysokowymiarowa $\rightarrow$ przestrzeń niskowymiarowa

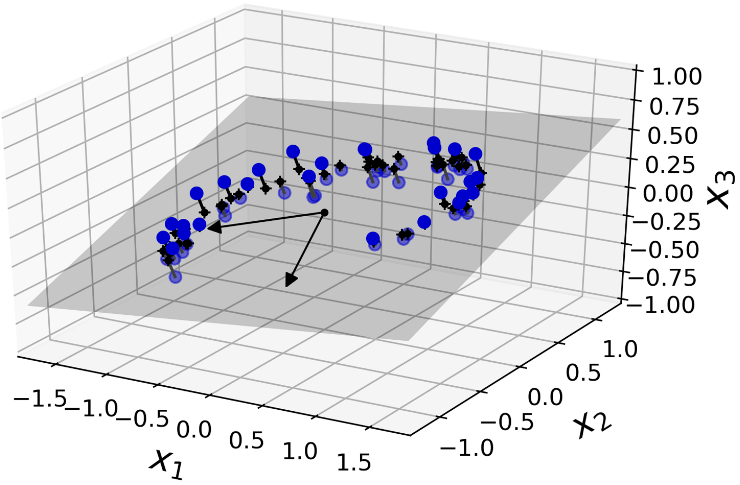


- zminejszenie liczby cech
  - przyspieszenie procesu uczenia/odpytywania
  - im więcej cech tym potrzeba znacznie więcej instancji
- wizualizacja

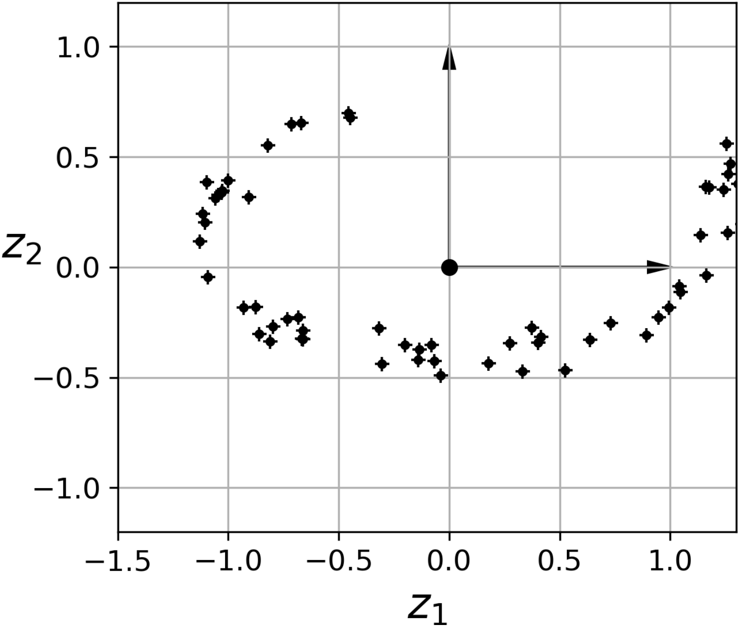
Projekcja

- Szczególnie dla rzadkich przestrzeni
  - instancje nie są równomiernie rozmieszczone
  - niektóre cechy są nieistotne
  - niektóre cechy są silnie skorelowane

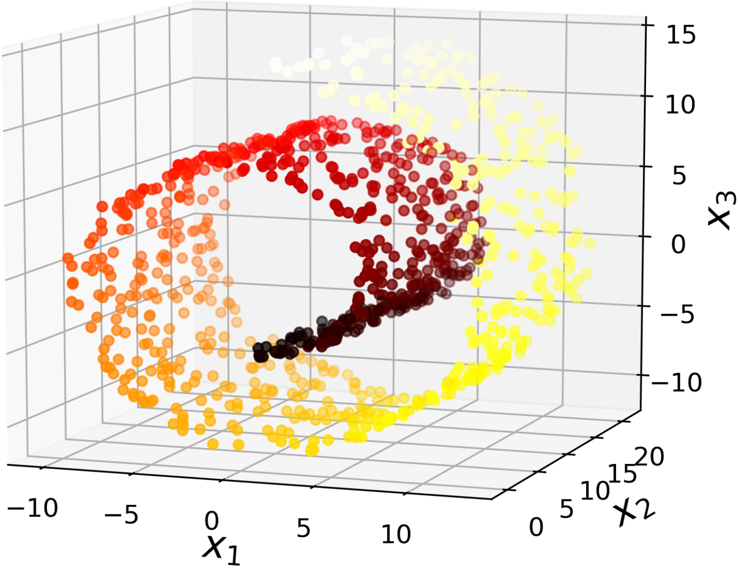
Projekcja, przykład



Projekcja, przykład, redukcja do 2D

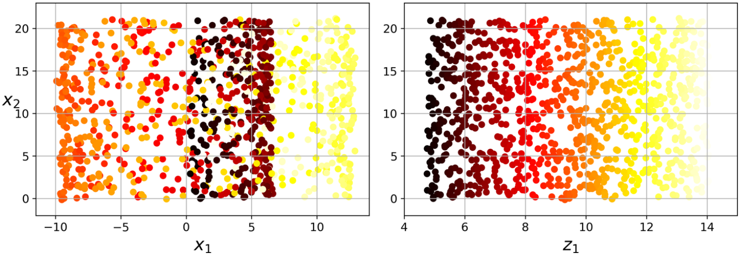


Rozmaitość, Manifold



- rozmaitość jest kształtem o niższej wymiarowości
- można go dopasować w przestrzeni o wyższej wymiarowości

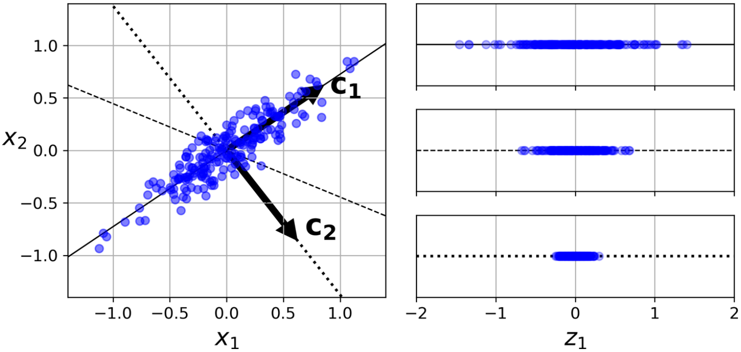
Manifold, przykład



Plan prezentacji

Analiza głównych składowych, Principal Component Analysis

- znajdź hiperpłaszczyznę najbliżej instancji
- rzutuj instancje na w/w hiperpłaszczyznę



- staraj się zachować jak najwięcej zmienności danych (instancji)
- minimalizacja odległości średniokwadratowej pomiędzy danymi oryginalnymi i projekcją
- Uwaga: dane muszą być centrowane

PCA, Singular Value Decomposition

- uśrednianie: standaryzacja zmiennych tak, aby średnia każdej zmiennej była 0
- macierz kowariancji
- wartości własne
- wektory własne,
- sortowanie w kierunku rosnącym wartości własnych (i wektorów własnych)
- wektor własny odpowiadający największej wartości nazywany jest składową główną
- pozostałe wektory - pozostałe składowe

Więcej informacji o podstawach statystyki...

PCA, SVD, przykład

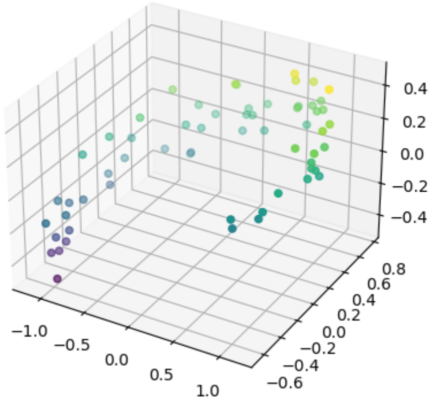
```python
import numpy as np
np.random.seed(4)
m = 60
w1, w2 = 0.1, 0.3
noise = 0.1

angles = np.random.rand(m) * 3 * np.pi / 2 - 0.5
X = np.empty((m, 3))
X[:, 0] = np.cos(angles) + \
    np.sin(angles)/2 + noise * np.random.randn(m) / 2
X[:, 1] = np.sin(angles) * 0.7 + \
    noise * np.random.randn(m) / 2
X[:, 2] = X[:, 0] * w1 + X[:, 1] * w2 + \
    noise * np.random.randn(m)
```

PCA, SVD, przykład

```python
import matplotlib.pyplot as plt
plt.figure()
ax = plt.axes(projection='3d')
ax.scatter3D(X[:,0], X[:,1], X[:,2], c=X[:,2])
f = 'pca-svd.png'
plt.savefig(f)
print(f)
```

PCA, SVD, wizualizacja



PCA, Scikit-Learn

- automatyczne centrowanie danych

```python
from sklearn.decomposition import PCA

pca = PCA(n_components=3)
X2D = pca.fit_transform(X)
print(pca.explained_variance_ratio_) # wsp.zmienności wym.
```

```
[0.84248607 0.14631839 0.01119554]
```

W praktyce chcemy mniej wymiarów:

```python
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
X2D = pca.fit_transform(X)
print(X.shape,'-->', X2D.shape)
print(pca.explained_variance_ratio_)
```

```
(60, 3) --> (60, 2)
[0.84248607 0.14631839]
```

PCA, ile wymiarów?

Jeżeli chcemy zachować 80% zmienności:

```python
from sklearn.decomposition import PCA

pca = PCA(n_components=0.8)
X2D = pca.fit_transform(X)
print(pca.explained_variance_ratio_)
```

```
[0.84248607]
```

PCA, transformacja odwrotna

```
X_recovered = pca.inverse_transform(X2D)
print(X_recovered.shape,'<--', X2D.shape)
```

```
(60, 3) <-- (60, 2)
```

Które cechy są najbardziej istotne?

```
from sklearn.decomposition import PCA
import numpy as np

pca = PCA(n_components=2)
X_reduced = pca.fit_transform(X)
c0_important_feature = np.argmax(abs(pca.components_[0]))
print(c0_important_feature)
print(pca.components_[0])
```

```
0
[-3.1, -3.0, 0.2]
```

- Należy przeprowadzić dla pozostałych składowych `components_`.
- Podejście naiwne, co jeżeli więcej niż jeden stary wymiar jest istotny dla nowego wymiaru?

Efekt uboczny: kompresja

```
from sklearn.datasets import fetch_openml
import time
mnist = fetch_openml('mnist_784', version=1, as_frame=False
mnist.target = mnist.target.astype(np.uint8)
X = mnist["data"]
y = mnist["target"]

start = time.time()
pca = PCA(n_components=0.95, svd_solver='full')
X_reduced = pca.fit_transform(X)
print(pca.n_components_)
print(np.sum(pca.explained_variance_ratio_))
print('time: ', time.time() - start)
```

```
154
0.950349970207861
time:  6.027488470077515
```

- wolne... $O(mn^2) + O(n^3)$, m – instancje, n – wymiary.

Efekt uboczny: kompresja

```
import pickle
X_recovered = pca.inverse_transform(X_reduced)
print(X.shape, ':' , len(pickle.dumps(X)))
print(X_reduced.shape, ':',
      len(pickle.dumps(X_reduced)))
print(X_recovered.shape, ':',
      len(pickle.dumps(X_recovered)))
```

```
(70000, 784) : 439040167
(70000, 154) : 86240166
(70000, 784) : 439040167
```

Kompresja, szybciej, randomized PCA

```python
from sklearn.datasets import fetch_openml
import time
mnist = fetch_openml('mnist_784', version=1, as_frame=False
mnist.target = mnist.target.astype(np.uint8)
X = mnist["data"]
y = mnist["target"]
start = time.time()
pca = PCA(n_components=154, svd_solver='randomized')
X_reduced = pca.fit_transform(X)
print(pca.n_components_)
print(np.sum(pca.explained_variance_ratio_))
print('time: ', time.time() - start)
```

```
154
0.9499886854344062
time:  4.3999693393707275
```

- trochę lepiej… $O(md^2) + O(d^3)$
- trzeba podać ile wymiarów: d.

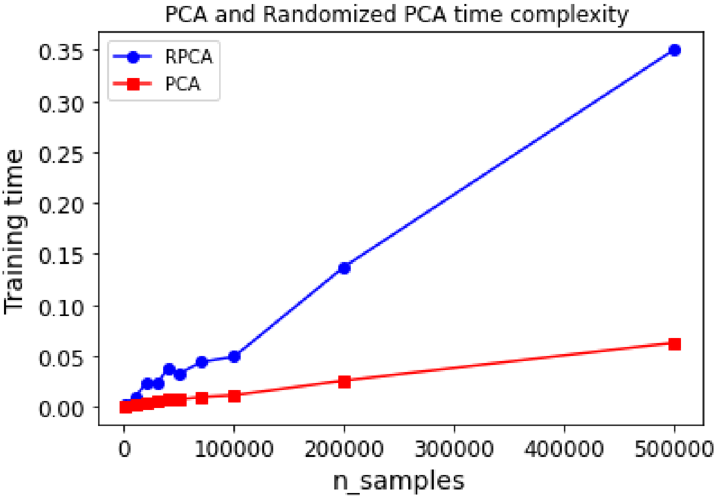
Kompresja, Incremental PCA

- minibatch
- out-of-core
- praca na strumieniach
- trzeba podać ile wymiarów

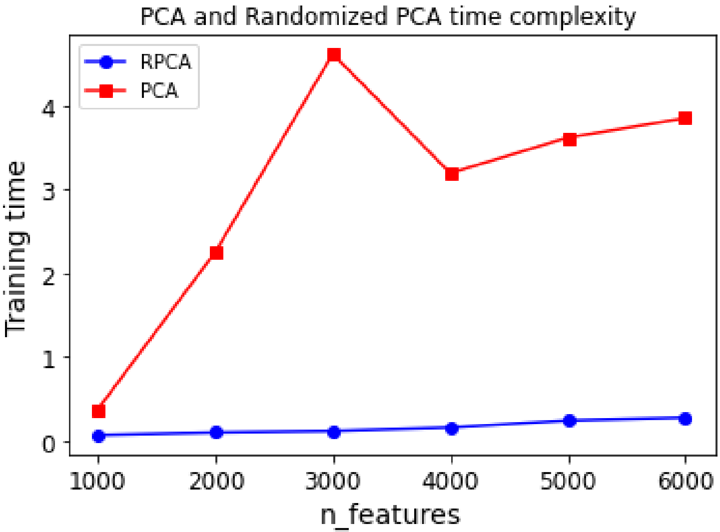
```
from sklearn.decomposition import IncrementalPCA
n_batches = 100
inc_pca = IncrementalPCA(n_components=154)
start = time.time()
for X_batch in np.array_split(X, n_batches):
    inc_pca.partial_fit(X_batch)
X_reduced = inc_pca.transform(X)
print(inc_pca.n_components_)
print(np.sum(inc_pca.explained_variance_ratio_))
print('time: ', time.time() - start)
```

```
154
0.9496651061456697
time:  12.997308731079102
```

Porównanie, zmienna liczba instancji



Porównanie, zmienna liczba cech



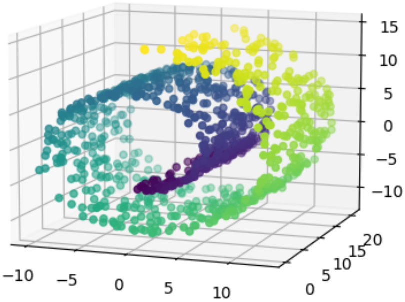
Plan prezentacji

Locally Linear Embedding, LLE

- Manifold

```python
from sklearn.datasets import make_swiss_roll
plt.figure()
plt.tight_layout()
X, t = make_swiss_roll(n_samples=1000, noise=0.2,
                                random_state=41)
ax = plt.axes(projection='3d')
ax.scatter3D(X[:,0], X[:,1], X[:,2], c=t)
ax.view_init(10, -70)
f = 'swiss-roll.png'
plt.savefig(f)
print(f)
```

LLE, swiss roll



LLE

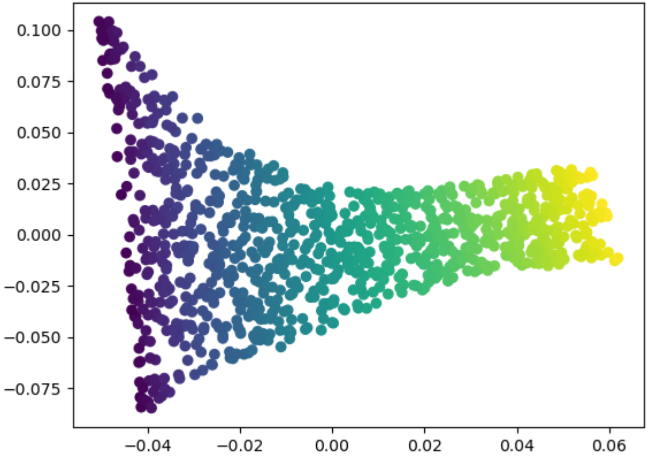
```
from sklearn.manifold import LocallyLinearEmbedding

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

- Uwaga: transformacja odwrotna nie jest trywialna.

```
plt.figure()
plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=t)
f = 'swiss-roll-unrolled.png'
plt.savefig(f)
print(f)
```

LLE



- lokalne odległości między instancja są zachowane, globalne nie. . .

LLE, algorytm

1. dla każdej instancji $x^{(i)}$ znajdź k najbliższych sąsiadów
2. wyznacz $x^{(i)}$ jako wartość funkcji powstałej z regresji liniowej sąsiadów
3. zmapuj $x^{(i)}$ na przestrzeń d-wymiarową, zachowując, jak tylko to możliwe w/w zależności liniowe

Plan prezentacji

Jaką technikę ML wybrać?

https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html