

Redukcja wymiarowości

Igor Wojnicki

April 21, 2023

Plan prezentacji

Redukcja Wymiarowości

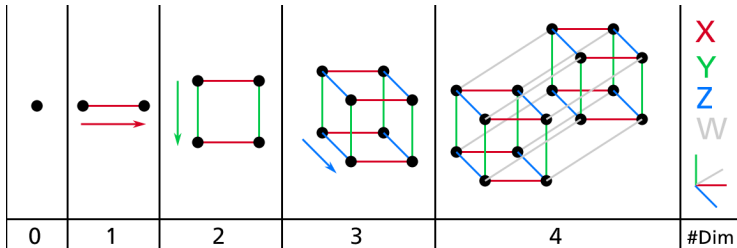
Projekcja

Rozmaitość, Manifold

Podsumowanie

Problem wymiarów

przestrzeń wysokowymiarowa \rightarrow przestrzeń niskowymiarowa



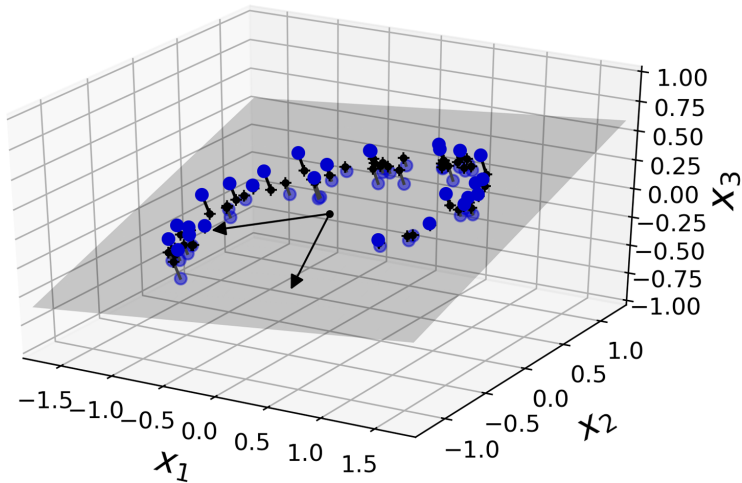
- ▶ zmniejszenie liczby cech
 - ▶ przyspieszenie procesu uczenia/odpytywania
 - ▶ im więcej cech tym potrzeba znacznie więcej instancji
- ▶ wizualizacja

Projekcja

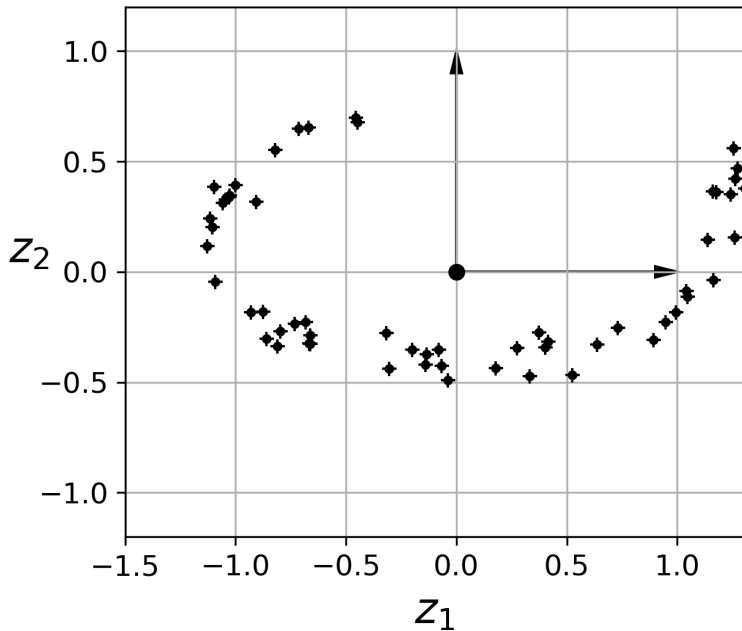
- ▶ Szczególnie dla rzadkich przestrzeni
 - ▶ instancje nie są równomiernie rozmieszczone
 - ▶ niektóre cechy są nieistotne
 - ▶ niektóre cechy są silnie skorelowane



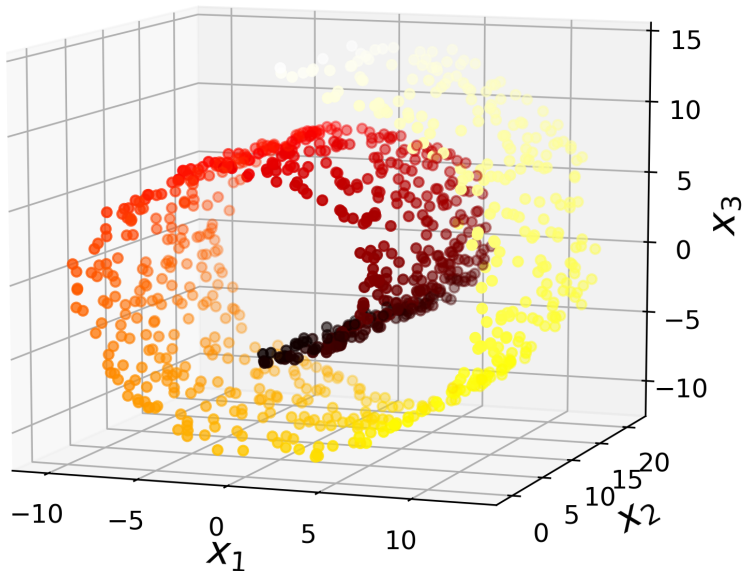
Projekcja, przykład



Projekcja, przykład, redukcja do 2D

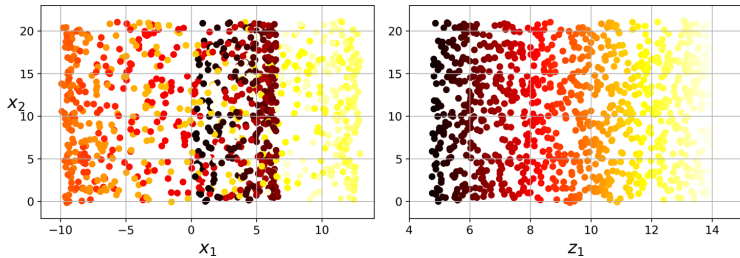


Rozmaitość, Manifold



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

Manifold, przykład



Plan prezentacji

Redukcja Wymiarowości

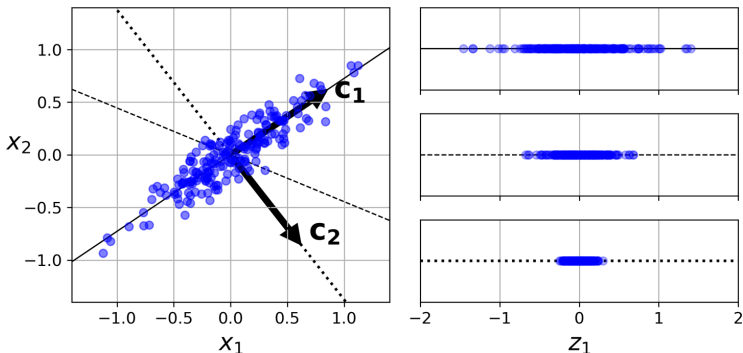
Projekcja

Rozmaitość, Manifold

Podsumowanie

Analiza głównych składowych, Principal Component Analysis

- ▶ znajdź hiperpłaszczyznę najbliższą 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

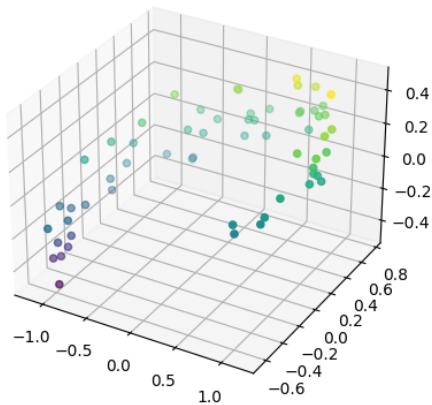
PCA, SVD, przykład

```
1  import numpy as np
2  np.random.seed(4)
3  m = 60
4  w1, w2 = 0.1, 0.3
5  noise = 0.1
6
7  angles = np.random.rand(m) * 3 * np.pi / 2 - 0.5
8  X = np.empty((m, 3))
9  X[:, 0] = np.cos(angles) + \
10      np.sin(angles)/2 + noise * np.random.randn(m) / 2
11  X[:, 1] = np.sin(angles) * 0.7 + \
12      noise * np.random.randn(m) / 2
13  X[:, 2] = X[:, 0] * w1 + X[:, 1] * w2 + \
14      noise * np.random.randn(m)
```

PCA, SVD, przykład

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

PCA, SVD, wizualizacja



PCA, Scikit-Learn

► automatyczne centrowanie danych

```
1 from sklearn.decomposition import PCA
2
3 pca = PCA(n_components=3)
4 X2D = pca.fit_transform(X)
5 print(pca.explained_variance_ratio_) # wsp.zmienności wym.
[0.84248607 0.14631839 0.01119554]
```

W praktyce chcemy mniej wymiarów:

```
1 from sklearn.decomposition import PCA
2
3 pca = PCA(n_components=2)
4 X2D = pca.fit_transform(X)
5 print(X.shape, '-->', X2D.shape)
6 print(pca.explained_variance_ratio_)
(60, 3) --> (60, 2)
[0.84248607 0.14631839]
```

PCA, ile wymiarów?

Jeżeli chcemy zachować 80% zmienności:

```
1 from sklearn.decomposition import PCA
2
3 pca = PCA(n_components=0.8)
4 X2D = pca.fit_transform(X)
5 print(pca.explained_variance_ratio_)
```

[0.84248607]

PCA, transformacja odwrotna

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

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

Które cechy są najbardziej istotne?

```
1  from sklearn.decomposition import PCA
2  import numpy as np
3
4  pca = PCA(n_components=3)
5  X_reduced = pca.fit_transform(X)
6  print(pca.explained_variance_ratio_)
7  c0_important_feature = np.argmax(abs(pca.components_[0]))
8  print(dataset.feature_names[c0_important_feature])
```

- ▶ należy przeprowadzić dla pozostałych składowych components_.

Efekt uboczny: kompresja

```
1  from sklearn.datasets import fetch_openml
2  import time
3  mnist = fetch_openml('mnist_784', version=1, as_frame=False)
4  mnist.target = mnist.target.astype(np.uint8)
5  X = mnist["data"]
6  y = mnist["target"]
7
8  start = time.time()
9  pca = PCA(n_components=0.95, svd_solver='full')
10 X_reduced = pca.fit_transform(X)
11 print(pca.n_components_)
12 print(np.sum(pca.explained_variance_ratio_))
13 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

```
1 import pickle
2 X_recovered = pca.inverse_transform(X_reduced)
3 print(X.shape, ':' , len(pickle.dumps(X)))
4 print(X_reduced.shape, ':',
5       len(pickle.dumps(X_reduced)))
6 print(X_recovered.shape, ':',
7       len(pickle.dumps(X_recovered)))
```

(70000, 784) : 439040167

(70000, 154) : 86240166

(70000, 784) : 439040167

Kompresja, szybciej, randomized PCA

```
1  from sklearn.datasets import fetch_openml
2  import time
3  mnist = fetch_openml('mnist_784', version=1, as_frame=False)
4  mnist.target = mnist.target.astype(np.uint8)
5  X = mnist["data"]
6  y = mnist["target"]
7  start = time.time()
8  pca = PCA(n_components=154, svd_solver='randomized')
9  X_reduced = pca.fit_transform(X)
10 print(pca.n_components_)
11 print(np.sum(pca.explained_variance_ratio_))
12 print('time: ', time.time() - start)
```

154

0.9499886854344062

time: 4.3999693393707275

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

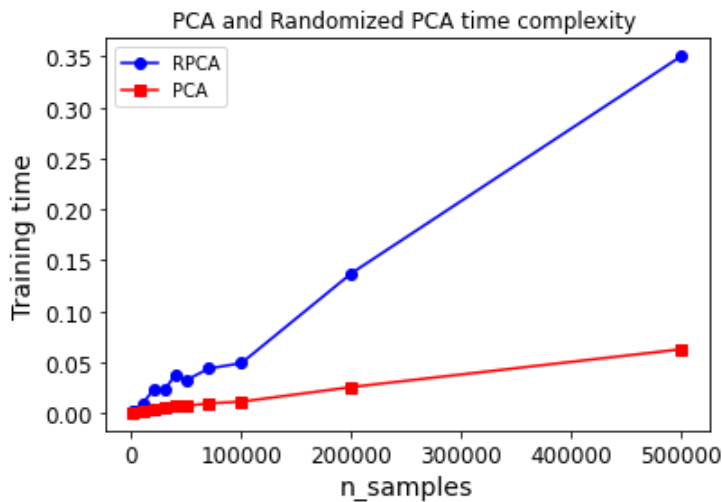
Kompresja, Incremental PCA

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

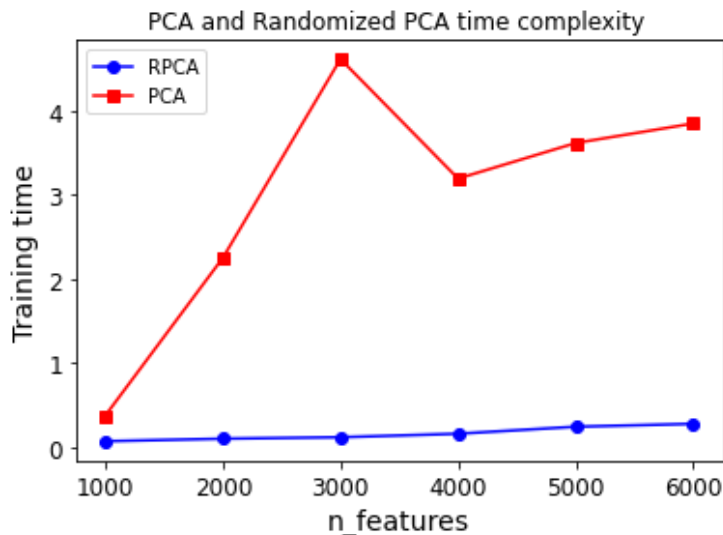
```
1  from sklearn.decomposition import IncrementalPCA
2  n_batches = 100
3  inc_pca = IncrementalPCA(n_components=154)
4  start = time.time()
5  for X_batch in np.array_split(X, n_batches):
6      inc_pca.partial_fit(X_batch)
7  X_reduced = inc_pca.transform(X)
8  print(inc_pca.n_components_)
9  print(np.sum(inc_pca.explained_variance_ratio_))
10 print('time: ', time.time() - start)

154
0.9496651061456697
time: 12.997308731079102
```

Porównanie, zmienna liczba instancji



Porównanie, zmienna liczba cech



Plan prezentacji

Redukcja Wymiarowości

Projekcja

Rozmaitość, Manifold

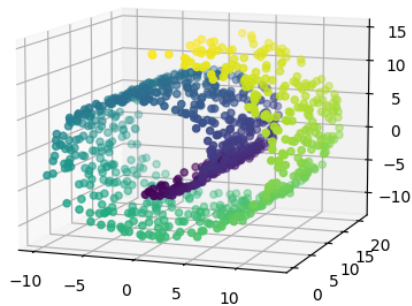
Podsumowanie

Locally Linear Embedding, LLE

► Manifold

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

LLE, swiss roll



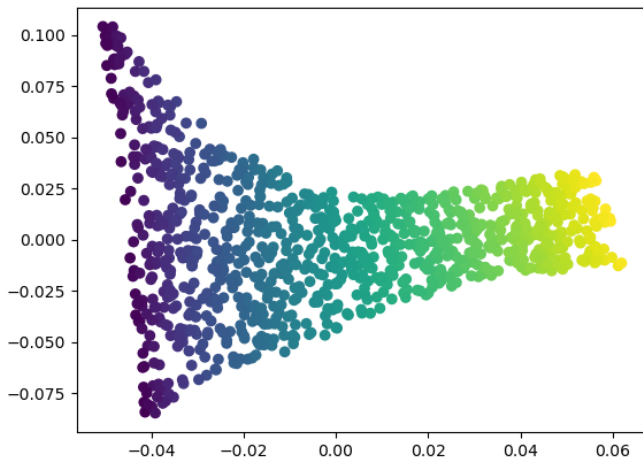
LLE

```
1  from sklearn.manifold import LocallyLinearEmbedding
2
3  lle = LocallyLinearEmbedding(n_components=2,
4                               n_neighbors=10,
5                               random_state=42)
6  X_reduced = lle.fit_transform(X)
```

► Uwaga: transformacja odwrotna nie jest trywialna.

```
1  plt.figure()
2  plt.scatter(X_reduced[:, 0], X_reduced[:, 1], c=t)
3  f = 'swiss-roll-unrolled.png'
4  plt.savefig(f)
5  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

Redukcja Wymiarowości

Projekcja

Rozmaitość, Manifold

Podsumowanie

Jaką technikę ML wybrać?

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

