Sprawozdanie 4 - SVD

Wojciech Smolarczyk, Wiktoria Zalińska

SVD - Singular Value Decomposition

Rozkład według wartości osobliwych polega na przedstawieniu dowolnej macierzy prostokątnej $A \in R^{n \times m}$ jako iloczynu trzech macierzy:

$$A = USV^T,$$

gdzie:

- $U \in R^{n \times n}$ - macierz ortogonalna ($U^T = U^{-1}$), zawierająca lewe wektory osobliwe - wektory własne AA^T,
- $S \in R^{n \times m}$ - macierz wartości osobliwych,
- $V \in R^{m \times m}$ - macierz ortogonalna ($V^T = V^{-1}$), zawierająca prawe wektory osobliwe - wektory własne $A^T A$.

Rozkład SVD isnieje dla każdej macierzy rzeczywistej - niezależnie od jej wymiaru i rangi.

Dzięki zastosowaniu SVD:

- można stabilnie rozwiązywać układy równań, nawet jeśli macierz jest osobliwa,
- wyznaczyć najlepiej dopasowane przybliżenie macierzy,
- analizować rangę, obraz i jądro macierzy.

Ranga i jądro macierzy

Ranga macierzy A, oznaczana jako $\text{rank}(A)$ - to liczba niezerowych wartości osobliwych σ_i.

Obraz macierzy A (czyli zbiór wszystkich możliwych wektorów $A\mathbf{x}$) jest generowany przez kolumny U odpowiadające niezerowym wartościom osobliwym.

Jądro (przestrzeń rozwiązań układu $A\mathbf{x} = 0$) jest związane z kolumnami V odpowiadającymi zerowym wartościom osobliwym.

Między wymiarem przestrzeni obrazu a jądra zachodzi następująca zależność:

$$dim(R(A)) + dim((N(A))) = m,$$

gdzie:

- $dim(R(A))$ - wymiar obrazu (ranga A)
- $dim((N(A)))$ - wymiar jądra - liczba niezależnych rozwiązań układu $A\mathbf{x} = 0$.

Kroki dekompozycji:

Krok 1. Rozważana macierz A ma postać:

```
A = np.array([[2, 2, 1, 3], [7, 8, 2, 8], [5, 0, 8, 7], [3, 5, 6, 3], [6, 9, 8, 4]])
n, m = A.shape
```

[[2 2 1 3]

[7 8 2 8]

[5 0 8 7]

[3 5 6 3]

[6 9 8 4]]

Rozkład przez AA^T

Krok 2. Macierz AA^T

```
AAT = np.dot(A, A.T)
```

AAT = [[18 56 39 31 50]

[56 181 107 97 162]

[39 107 138 84 122]

[31 97 84 79 123]

[50 162 122 123 197]]

AAT shape = (5, 5)

Otrzymano macierz $n \times n$ (u nas 5×5).

Krok 3. Wartości i wektory własne macierzy AA^T

```
eigenvalues, eigenvectors = np.linalg.eig(AAT)

idx = np.argsort(eigenvalues)[::-1]
eigenvalues = eigenvalues[idx]
eigenvectors = eigenvectors[:, idx]
U = eigenvectors
```

Eigenvalues = [5.27198727e+02 5.22965267e+01 3.29123797e+01 5.92366606e-01 -2.64827854e-16]

Eigenvectors =

[[0.17354228 0.06750887 0.23303841 -0.4040119 -0.86475032]

[0.54275123 0.5240643 0.58684982 0.03444987 0.29188733]

[0.43235249 -0.83173678 0.31478192 0.14364195 0.03955483]

[0.37272532 -0.07348099 -0.39457998 -0.77167806 0.3232584]

[0.5911441 0.15366728 -0.58865896 0.46847364 -0.24687667]]

Shape of eigenvectors (U) = (5, 5)

Otrzymaliśmy macierz U o wymairach $n \times n$.

Krok 4. Macierz diagonalna S taka że $S_{ii} = \sqrt{\lambda_i}$

```
S = np.zeros((n, m))
for i in range(min(n, m)):
   S[i, i] = np.sqrt(eigenvalues[i]) if eigenvalues[i] > 1e-10 else 0.0
```

S = [[22.9608085 0. 0. 0.]

[0. 7.23163375 0. 0.]

[0. 0. 5.73693121 0.]

[0. 0. 0. 0.76965356]

[0. 0. 0. 0.]]

S shape = (5, 4)

Krok 5. Obliczenie macierzy V z własności: $V = A^T U S^{-1}$

```
S_inv = np.zeros_like(S)
for i in range(m):
    if S[i, i] > 1e-10:
        S_inv[i, i] = 1.0 / S[i, i]

V = A.T @ U @ S_inv
```

V.T =

[[0.47790765 0.51709929 0.50883963 0.49527247]

[0.047893 0.73885555 -0.65680932 -0.14282946]

[0.24965289 -0.36778463 -0.54938051 0.70752311]

[0.84082339 -0.22679343 -0.08868373 -0.48344179]]

Krok 6. Rekonstrukcja macierzy A

```
reconstructed_A_v1 = U @ S @ V.T
```

array([[2.00000000e+00, 2.00000000e+00, 1.00000000e+00, 3.00000000e+00],
[7.00000000e+00, 8.00000000e+00, 2.00000000e+00, 8.00000000e+00],
[5.00000000e+00, 5.86468211e-15, 8.00000000e+00, 7.00000000e+00],
[3.00000000e+00, 5.00000000e+00, 6.00000000e+00, 3.00000000e+00],
[6.00000000e+00, 9.00000000e+00, 8.00000000e+00, 4.00000000e+00]])

Rozkład przez $A^T A$

Krok 7

Proszę obliczyć i wypisać/narysować macierz ATA(mxm)

```
ATA = np.dot(A.T, A)
```

```
print(ATA)
```

Krok 8

Proszę (używając stosownej biblioteki) policzyć wartości i i wektory własne Vi macierzy ATA.

```
lambdas, V = np.linalg.eig(ATA)
print("\nStep 8: Eigenvalues ( _i) of A^T A:")
print(lambdas)
print("\nEigenvectors (V_i) of A^T A (columns):")
print(V)
```

Eigenvalues (_i) of A^T A:

[430.73196147 59.91931837 6.26331361 20.08540655]

Eigenvectors (V_i) of A^T A (columns):

[[0.61815934 0.29989507 0.57592274 0.44300674]

[0.44206838 -0.10849764 0.24128459 -0.85707967]

[0.56014362 -0.65084557 -0.45039358 0.24450931]

[0.32968729 0.68897842 -0.63815387 -0.09682286]]

Krok 9

```python
print("\n9. Macierz V^T:\n", V.T)


S = np.zeros((m, m))
for i in range(m):
    S[i, i] = np.sqrt(lambdas[i]) if lambdas[i] > 1e-10 else 0.0
print("\nStep 9: Diagonal S (manual):\n", S)
```

Macierz V^T:
[[0.61815934 0.44206838 0.56014362 0.32968729]
[0.29989507 -0.10849764 -0.65084557 0.68897842]
[0.57592274 0.24128459 -0.45039358 -0.63815387]
[0.44300674 -0.85707967 0.24450931 -0.09682286]]

Diagonal S :
[[20.75408301 0. 0. 0.]
[0. 7.74075696 0. 0.]
[0. 0. 2.50266131 0.]
[0. 0. 0. 4.48167452]]

Krok 10 i 11

```python
S_inv = np.zeros_like(S)
for i in range(m):
    if S[i, i] > 1e-10:
        S_inv[i, i] = 1.0 / S[i, i]

U = A @ V @ S_inv
print("\nStep 11: U = [U_1 U_2 ... U_m]:\n", U)
```

U = [U_1 U_2 ... U_m]:
[[0.17681634 0.2323913 -0.29186551 -0.19503985]
[0.5599596 0.70295717 -0.01769403 -0.90170635]
[0.47603918 0.14411507 -2.07403694 0.77947387]
[0.40545003 -0.19131735 -0.67233703 -0.3971263]
[0.64987066 -0.21031008 -0.21124164 -0.77803363]]

Krok 12

```python
reconstructed_A = U @ S @ V.T
print(
    "\nStep 12: Reconstructed A (should match original A):\n",
```

```
    "Using AAT decomposition:\n",
    reconstructed_A_v1,
    "\n\n",
    "Using ATA decomposition:\n",
    reconstructed_A,
    "\n\n",
    "Original A:\n",
    A,
)
```

Step 12: Reconstructed A (should match original A): Using AAT decomposition:

[[2.00000000e+00 2.00000000e+00 1.00000000e+00 3.00000000e+00]

[7.00000000e+00 8.00000000e+00 2.00000000e+00 8.00000000e+00]

[5.00000000e+00 5.86468211e-15 8.00000000e+00 7.00000000e+00]

[3.00000000e+00 5.00000000e+00 6.00000000e+00 3.00000000e+00]

[6.00000000e+00 9.00000000e+00 8.00000000e+00 4.00000000e+00]]

Using ATA decomposition: [[2.00000000e+00 2.00000000e+00 1.00000000e+00 3.00000000e+00]

[7.00000000e+00 8.00000000e+00 2.00000000e+00 8.00000000e+00]

[5.00000000e+00 -1.77635684e-15 8.00000000e+00 7.00000000e+00]

[3.00000000e+00 5.00000000e+00 6.00000000e+00 3.00000000e+00]

[6.00000000e+00 9.00000000e+00 8.00000000e+00 4.00000000e+00]]

Original A:

[[2 2 1 3]

[7 8 2 8]

[5 0 8 7]

[3 5 6 3]

[6 9 8 4]]

Krok 13

Obliczmy wykorzystując wzory:

rankA = dimR(A)

dimR(A) + dimN (A) = m

```python
rank_A = np.sum(lambdas > 1e-10)
nullity_A = m - rank_A
print("\n Dimensions:")
print(f"dim(R(A)) = rank(A) = {rank_A}")
print(matrix_rank(A))

print(f"dim(N(A)) = nullity(A) = {nullity_A}")
```

Dimensions:
dim(R(A)) = rank(A) = 4
4
dim(N(A)) = nullity(A) = 0