

POLITECHNIKA WROCŁAWSKA

INTELIGENCJA OBLICZENIOWA I JEJ ZASTOSOWANIA

Ćwiczenie 2
Metody redukcji wymiarowości
– nieujemna faktoryzacja macierzy
i dekompozycje tensorów

Autorzy:

Paweł ANDZIUL 200648
Robert CHOJNACKI 200685
Marcin SŁOWIŃSKI 200638

Prowadzący:

dr hab. inż. Rafał ZDUNEK

9 czerwca 2017

Spis treści

1	Zadanie 1	2
1.1	Algorytm ALS	2
1.2	Algorytm MUE	2
1.3	Algorytm HALS	2
1.4	Realizacja	3
1.5	Wyniki	5
2	Zadanie 2	6
3	Zadanie 3	6
3.1	Opis metody	6
3.2	Algorytm	6
3.3	Realizacja	6
3.4	Wyniki	9
4	Podsumowanie	15

1 Zadanie 1

Wygenerować faktory $A = [a_{ij}] \in R_+^{I \times J}$ i $X = [x_{jt}] \in R_+^{J \times T}$, gdzie $a_{ij} = \max(0, \check{a}_{ij})$ i $x_{jt} = \max(0, \check{x}_{jt})$ oraz $\check{a}_{ij}, \check{x}_{jt} \sim N(0, 1)$ (rozkład normalny). Wygeneruj syntetyczne obserwacje $Y = AX$ dla $I = 100, T = 1000, J = 10$. Stosując wybrane algorytmy NMF (ALS, MUE, HALS) wyznacz estymowane faktory \hat{A} i \hat{X} oraz unormowany błąd residualny w funkcji iteracji naprzemiennych. Oceń jakość estymacji stosując miary MSE (ang. Mean-Squared Error) lub SIR (ang. Signal-to-Interference Ratio).

1.1 Algorytm ALS

Algorytm ALS polega na N-krotnym powtórzeniu pętli, której zadaniem jest obliczenie tensorów A oraz X . Wynik obliczenia jednego tensora wpływa na wynik drugiego, co dla kolejnych iteracji zmniejsza różnice między obrazem właściwym a zredukowanym. Algorytm bazuje na wzorach 1 oraz 2.

$$\nabla_A D(Y|AX) = (AX - Y)X^T = 0 \Rightarrow AX X^T = Y X^T \Rightarrow A = Y X^T (X X^T)^{-1} \quad (1)$$

$$\nabla_X D(Y|AX) = A^T (AX - Y) = 0 \Rightarrow A^T AX = A^T Y \Rightarrow X = (A^T A)^{-1} A^T Y \quad (2)$$

Otrzymane tensory po każdej z iteracji przemnażane są przez siebie otrzymując obraz zredukowany, który następnie jest przyrównywany do oryginału oznaczonego jako Y .

1.2 Algorytm MUE

Algorytm MUE działa podobnie do algorytmu ALS. Zmieniają się w nim wzory odpowiadające za poszczególne tensory.

$$a_{ij} = a_{ij} \frac{[Y X^T]_{ij}}{[A X X^T]_{ij}} \quad (3)$$

$$x_{jt} = x_{jt} \frac{[A^T Y]_{jt}}{[A^T A]_{jt}} \quad (4)$$

Zaletą algorytmu MUE jest niski koszt obliczeniowy. Niestety, jest on obciążony kosztem w postaci powolnej zbieżności, co oznacza konieczność wykonania większej ilości obliczeń, które mogą zanegować jego zalety.

1.3 Algorytm HALS

Algorytm HALS od algorytmu ALS różni się iterowaniem osobno po wierszach i kolumnach.

$$a_j = [a_j + \frac{[Y X^T]_{*j} - A [X X^T]_{*j}}{[X X^T]_{jj}}]_+ \quad (5)$$

$$x_j = [x_j + \frac{[A^T Y]_{j*} - [A^T A]_{j*} X}{[A^T A]_{jj}}]_+ \quad (6)$$

Algorytm ten charakteryzuje się najmniejszym błędem residualnym osiąganym już przy niskiej ilości iteracji.

1.4 Realizacja

Na poniższych listingach zamieszczono praktyczną implementację algorytmów w środowisku MATLAB.

Listing 1: Skrypt wywołujący realizacje w środowisku MATLAB

```
1 clc;
2 clear all;
3 close all;
4
5 % dane oryginalne
6 I = 100; T = 1000; J = 10;
7 Aw = max(0,randn(I,J));
8 Xw = max(0,randn(J,T));
9
10 Y = Aw*Xw;
11
12 % inicjalizacja
13 A = rand(size(Y,1),J);
14 X = rand(J,size(Y,2));
15
16 A1 = A; A2 = A; A3 = A;
17 X1 = X; X2 = X; X3 = X;
18
19 MaxIter = 300;
20
21 [A1,X1,res1] = skrypt_zad1_nmf_als(A1,X1,Y,MaxIter);
22 [A2,X2,res2] = skrypt_zad1_nmf_mue(A2,X2,Y,MaxIter);
23 [A3,X3,res3] = skrypt_zad1_nmf_hals(A3,X3,Y,J,MaxIter);
24
25 figure
26 semilogy(res1);
27 hold on;
28 semilogy(res2);
29 semilogy(res3);
30 legend('ALS', 'MUE', 'HALS');
31 hold off;
32 grid on;
33
34 SIR_A1 = CalcSIR(A',A1');
35 SIR_A2 = CalcSIR(A',A2');
36 SIR_A3 = CalcSIR(A',A3');
37 %%
38 figure;
39 hold on
40 title('Wartosci SIR');
41 plot(1:100, SIR_A1, 1:100, SIR_A2, 1:100, SIR_A3);
42 xlim([1 100]);
43 legend('ALS', 'MUE', 'HALS', 'Location','southeast','Orientation','vertical')
44 ylabel('SIR')
45 hold off
```

Listing 2: Algorytm ALS

```

1 function [A,X,res] = skrypt_zad1_nmf_als(A,X,Y,N)
2     for k = 1:N
3         % obliczanie A
4         A = max(0,Y*X'*inv(X*X'));
5         A = A*diag(1./sum(A,1));
6
7         % obliczanie X
8         X = max(0,inv(A'*A)*A'*Y);
9
10        % blad residualny
11        res(k) = norm(Y - A*X,'fro')/norm(Y,'fro');
12    end
13 end

```

Listing 3: Algorytm MUE

```

1 function [A,X,res] = skrypt_zad1_nmf_mue(A,X,Y,N)
2     for k = 1:N
3         % obliczanie A
4         s1 = A.*(Y*X');
5         s2 = (A*X*X');
6         A = max(0, s1./s2);
7         A = A*diag(1./sum(A,1));
8
9         % obliczanie X
10        s1 = X.*(A'*Y);
11        s2 = A'*A*X;
12        X = max(0, s1./s2);
13
14        % blad residualny
15        res(k) = norm(Y - A*X, 'fro')/norm(Y,'fro');
16    end
17 end

```

Listing 4: Algorytm HALS

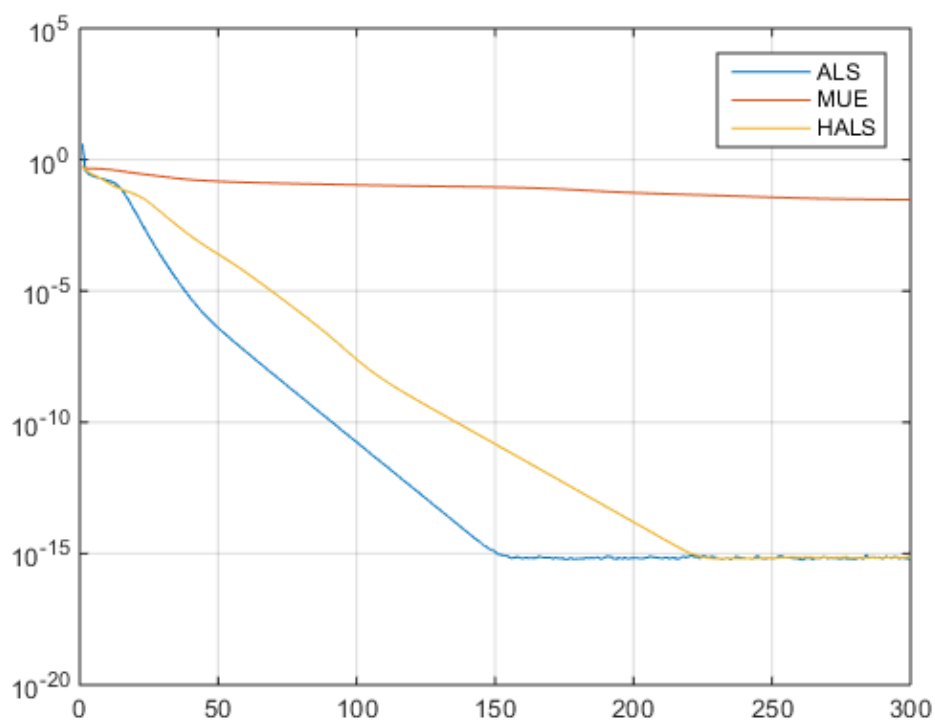
```

1 function [A,X,res] = skrypt_zad1_nmf_hals(A,X,Y,J,MaxIter)
2     for k = 1:MaxIter
3         for j = 1:J % obliczanie A - po kolumnach
4             YXp = Y*X';
5             XXp = X*X';
6             A(:,j) = max(0, A(:,j)+(YXp(:,j)-A*XXp(:,j))/(XXp(j,j)+eps));
7             A = A*diag(1./sum(A,1));
8         end
9         for j = 1:J % obliczanie X - po wierszach
10            ApY = A'*Y;
11            ApA = A'*A;
12            X(j,:) = max(0, X(j,:)+(ApY(j,:)-ApA(j,:)*X)/(ApA(j,j)+eps));
13        end
14        res(k) = norm(Y - A*X, 'fro')/norm(Y,'fro'); % blad residualny
15    end
16 end

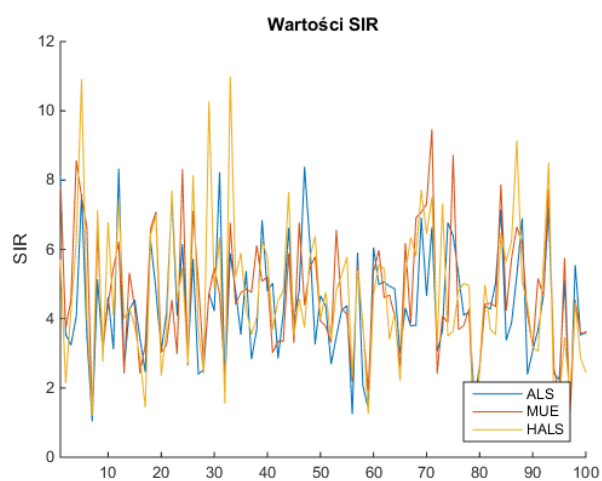
```

1.5 Wyniki

Na ilustracji 1 zamieszczono graficzne porównanie przebiegu optymalizacji kolejno algorytmami ALS, MUE oraz HALS.



Rysunek 1: Wykres ilustrujący przebieg optymalizacji ALS, MUE oraz HALS



Rysunek 2: Wykres SIR

2 Zadanie 2

Wygenerować faktory $U(1) = [u_{ij}^{(1)}] \in R_+^{I_1 \times J}$, $U(2) = [u_{ij}^{(2)}] \in R_+^{I_2 \times J}$, $U(3) = [u_{ij}^{(3)}] \in R_+^{I_3 \times J}$, gdzie $[u_{ij}^{(1)}] = \max(0, \check{u}_{ij}^{(1)})$, $[u_{ij}^{(2)}] = \max(0, \check{u}_{ij}^{(2)})$, $[u_{ij}^{(3)}] = \max(0, \check{u}_{ij}^{(3)})$ oraz $\check{u}_{ij}^{(1)}, \check{u}_{ij}^{(2)}, \check{u}_{ij}^{(3)} \sim N(0, 1)$ (rozkład normalny). Wygeneruj syntetyczne obserwacje Y dla $I_1 = 10$, $I_2 = 20$, $I_3 = 30$, $J = 5$. Stosując wybrane algorytmy NTF (np. ALS) wyznacz estymowane faktory $\hat{U}^{(1)}, \hat{U}^{(2)}, \hat{U}^{(3)}$ oraz unormowany błąd residualny w funkcji iteracji naprzemiennych. Oceń jakość estymacji stosując miary MSE (ang. Mean-Squared Error) lub SIR (ang. Signal-to-Interference Ratio).

3 Zadanie 3

Obrazy twarzy z bazy ORL (lub podobnej) przedstaw za pomocą tensora $Y \in R^{I_1 \times I_2 \times I_3}$, gdzie I_3 jest liczbą obrazów. Rozdziel obrazy na zbiory trenujący i testujący według odpowiedniej zasady, np. 5-folds CV i utwórz odpowiednie tensory trenujący Y_r i testujący Y_t . Tensor trenujący poddaj dekompozycji CP (np. algorytmem ALS) oraz HOSVD dla $J = 4, 10, 20, 30$. Pogrupować obrazy stosując metodę k-średnich dla faktora $\hat{U}^{(3)}$. Badania przeprowadzić dla różnej liczby grup. Porównać dokładność grupowania z metodą PCA (z poprzedniego ćwiczenia). Następnie dokonaj projekcji obrazów z tensora Y_t na podprzestrzeń cech generowaną czynnikami otrzymanymi z Y_r . Dokonaj klasyfikacji obrazów w przestrzeni cech w $\hat{U}^{(3)}$ za pomocą klasyfikatora k-NN. Porównać efekty klasyfikacji różnymi metodami (np. PCA, CP, HOSVD).

3.1 Opis metody

HOSVD jest szczególnym przypadkiem modelu dekompozycji Tuckera. Kolumny każdej z macierzy czynnikowych (faktorów) są wzajemnie ortogonalne, a tensor rdzeniowy \mathcal{G} nie jest superdiagonalny. Dokonujemy kolejno matrycyzacji względem n-tego modu.

3.2 Algorytm

Wejściem algorytmu HO-SVD są: tensor danych $Y \in R^{I_1 \times I_2 \times I_3}$ oraz rzędy faktoryzacji $J = [J_1, \dots, J_N]$, natomiast wyjściem: estymowane wektory $U^{(n)}$ oraz tensor rdzeniowy \mathcal{G} .

Pseudokod 1 Algorytm HO-SVD

- 1: **for** $n = 1, \dots, N$ **do**
 - 2: $Y_{(n)} \leftarrow \text{unfolding}(\mathcal{Y}, n) \in R^{I_n \times \prod_{p \neq n} I_p}$
 - 3: $U_{(n)} \leftarrow \text{eigs}(Y_{(n)}(Y_{(n)})^T, J_n)$
 - 4: Wyznacz tensor \mathcal{G}
-

3.3 Realizacja

Na kolejnych listingach zamieszczono implementację w środowisku MATLAB. Listing 5 stanowi podstawowy potok przetwarzania od wczytania danych po wyświetlenie wyników pomiarów. Na listingu 6 zamieszczono implementację metody HOSVD. W celu przeprowadzenia klasyfikacji kNN dokonano podziału zbioru wejściowego przy pomocy *cvpartition* 5-folds (podzielono 80 zdjęć na dwa zbiory składające się odpowiednio z 64 i 16 obrazów).

Listing 5: Podstawowy skrypt z realizacją w środowisku MATLAB

```

1  clc;
2  clear;
3  close all;
4  load('FaceData_56_46.mat');
5
6  Persons = 8;
7  ImagesPerPerson = 10;
8  nOfImages = Persons*ImagesPerPerson;
9
10 % wczytanie danych do tensora
11 P = zeros(1,nOfImages);
12 Y=zeros(56,46,nOfImages);
13 img_index = 1;
14 for p=(1:Persons)
15     for i=(1:ImagesPerPerson)
16         P(img_index) = p;
17         Y(:,:,img_index) = FaceData(p, i).Image;
18         img_index = img_index + 1;
19     end
20 end
21 P = P';
22
23 figure;
24 subtitle('Twarze oryginalne');
25 for i=(1:nOfImages)
26     subplot(Persons, ImagesPerPerson, i);
27     imagesc(Y(:,:,i));
28     title(i)
29     colormap gray;
30     set(gca,'XtickLabel',[], 'YtickLabel',[]);
31 end
32
33 % rozdzielenie na dwa zbiory (5-folds CV)
34 CV = cvpartition(P,'kfold',5);
35 train_idx = CV.training(1);
36 test_idx = CV.test(1);
37
38 % utworzenie tensorow trenujacego i testowego
39 Y_train = Y(:,:,train_idx);
40 Y_test = Y(:,:,test_idx);
41 Class_train_idx = P(train_idx);
42 Class_test_idx = P(test_idx);
43
44 J_serie = [4 10 20 30];
45
46 res_time_hosvd = zeros(1,length(J_serie));
47 res_time_kmeans = zeros(1,length(J_serie));
48 res_time_knn = zeros(1,length(J_serie));
49 res_acc_kmeans = zeros(1,length(J_serie));
50 res_acc_knn = zeros(1,length(J_serie));
51 res_rands_kmeans = zeros(1,length(J_serie));
52 res_rands_knn = zeros(1,length(J_serie));
53 res_delta = zeros(1,length(J_serie));
54 res_groups_kmeans = [];

```



```

55
56 for J_current=(1:length(J_serie))
57     J(1:3) = J_serie(J_current);
58
59     % dekompozycja hosvd (pod kmeansa)
60     tic
61     [A, B, C, G, Y_hat] = skrypt_zad3_hosvd(Y, J);
62     res_time_hosvd(J_current) = toc;
63
64     figure;
65     suptitle(sprintf('Tworzenie zredukowane J=%d (HOSVD)', J_serie(J_current)));
66     for i=(1:nOfImages)
67         subplot(Persons, ImagesPerPerson, i);
68         imagesc(Y_hat(:,:,i));
69         title(i)
70         colormap gray;
71         set(gca, 'XtickLabel', [], 'YtickLabel', []);
72     end
73
74     % grupowanie metoda ksrednich dla faktora U^(3) - stala liczba grup (ilosc
       osob)
75     tic
76     kmeans_result = kmeans(C, Persons);
77     res_time_kmeans(J_current) = toc;
78     res_groups_kmeans = [res_groups_kmeans kmeans_result];
79     [res_acc_kmeans(J_current), res_rands_kmeans(J_current), ~] = AccMeasure(P,
       kmeans_result');
80
81     % dekompozycja hosvd
82     [Ar, Br, Cr, Gr, Yr_hat] = skrypt_zad3_hosvd(Y_train, J);
83
84     % projekcja
85     Y3 = reshape(permute(Y_test, [3 1
       2]), size(Y_test,3), size(Y_test,1)*size(Y_test,2));
86     G3 = reshape(permute(Gr, [3 1 2]), [J(3), J(1)*J(2)]);
87     Ct = Y3*pinv(double(G3)*(kron(Br,Ar))');
88     Ct = Ct.*repmat(1./sqrt(sum(Ct.^2,2)+eps), 1, size(Ct,2));
89
90     % klasyfikacja w przestrzeni cech U^(3)
91     tic
92     mdl_class = fitcknn(Cr, Class_train_idx, 'NumNeighbors', 1);
93     prediction = predict(mdl_class, Ct);
94     res_time_knn(J_current) = toc;
95     [res_acc_knn(J_current), res_rands_knn(J_current), ~] =
       AccMeasure(prediction, Class_test_idx');
96
97     % dokladnosc klasyfikacji (podobnie jak w AccMeasure)
98     res_delta(J_current) = 100*(length(find((prediction -
       Class_test_idx)==0))/length(Class_test_idx));
99
100 end
101 %%
102 figure;
103 hold on
104 title('Czas przetwarzania dla roznych wartosci J');
105 plot(J_serie, res_time_hosvd, J_serie, res_time_kmeans, J_serie, res_time_knn);

```

```

106 xlim([4 30]);
107 legend('hosvd','kmeans','knnclassify',
        'Location','southeast','Orientation','vertical')
108 xlabel('J')
109 ylabel('czas przetwarzania [s]')
110 hold off

```

Listing 6: Funkcja realizująca algorytm HOSVD

```

1 function [ A, B, C, G, Y_hat ] = skrypt_zad3_hosvd( Y, J )
2
3     DimY = size(Y);
4
5     % unfolding
6     Y1 = reshape(Y,DimY(1),DimY(2)*DimY(3));
7     Y2 = reshape(permute(Y,[2 1 3]),DimY(2),DimY(1)*DimY(3));
8     Y3 = reshape(permute(Y,[3 1 2]),DimY(3),DimY(1)*DimY(2));
9
10    % dekompozycja tensorow
11    [E1,~] = eig(Y1*Y1');
12    A = fliplr(E1(:,DimY(1)-J(1)+1:DimY(1)));
13
14    [E2,~] = eig(Y2*Y2');
15    B = fliplr(E2(:,DimY(2)-J(2)+1:DimY(2)));
16
17    [E3,~] = eig(Y3*Y3');
18    C = fliplr(E3(:,DimY(3)-J(3)+1:DimY(3)));
19
20    G = ntimes(ntimes(ntimes(Y,A',1,2),B',1,2),C',1,2); % core tensor
21    Y_hat = ntimes(ntimes(ntimes(G,A,1,2),B,1,2),C,1,2); % tensor 3-way
22
23    C = C.*repmat(1./sqrt(sum(C.^2,2)+eps),1,size(C,2));
24
25 end

```

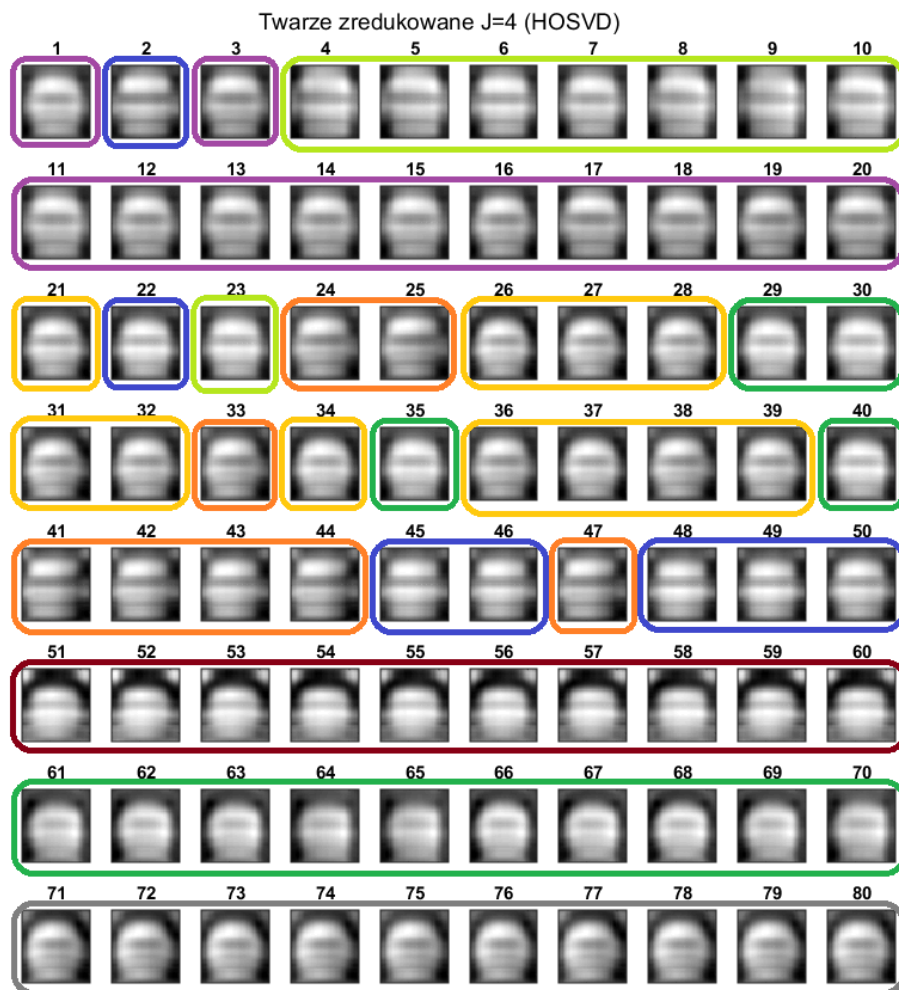
3.4 Wyniki

W niniejszym punkcie zamieszczono wyniki dla dekompozycji z wykorzystaniem metody HOSVD. Badaniu poddano 80 obrazów twarzy (ilustracja 3) z bazy Uniwersytetu Cambridge [7].



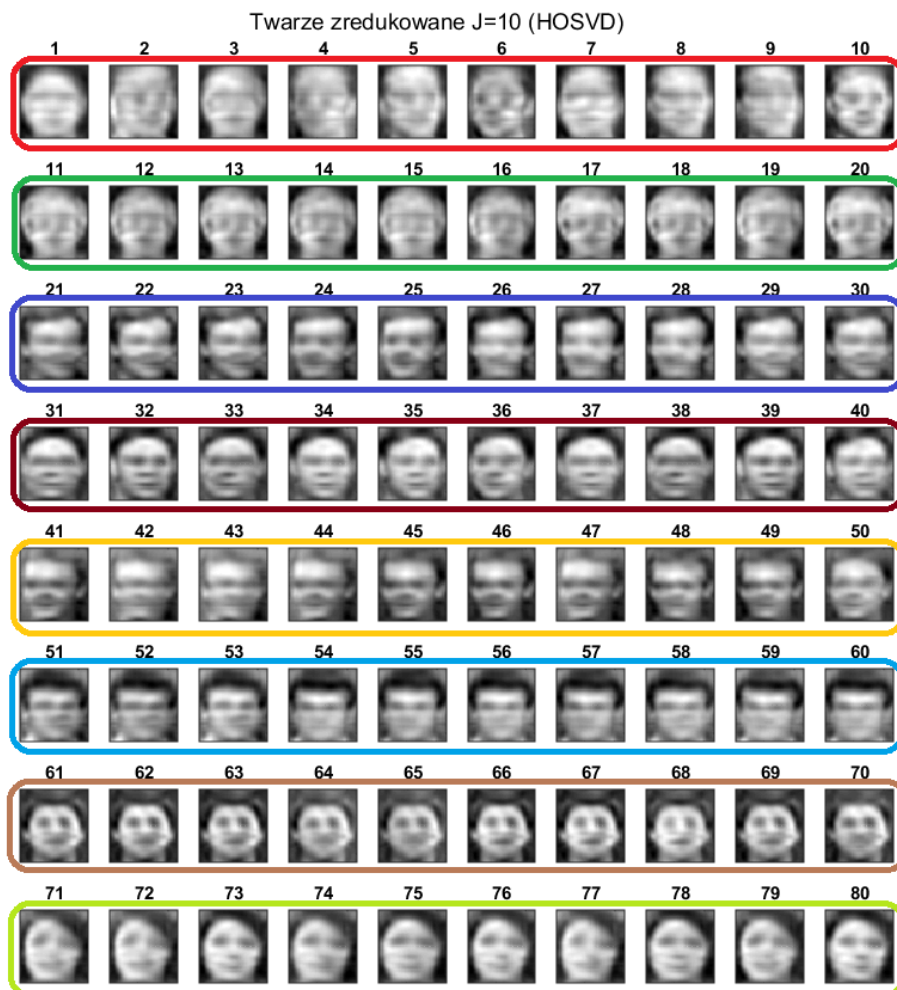
Rysunek 3: Twarze wykorzystane podczas testów

Ilustracja 4 przedstawia wynik grupowania metodą k-średnich dla $J=4$ oraz liczby grup równej liczbie osób wynoszącej 8.



Rysunek 4: Twarze zredukowane $J=4$ (HOSVD)

Rysunek 5 przedstawia wynik grupowania metodą k-średnich dla $J=10$ oraz liczby grup równej liczbie osób wynoszącej 8.



Rysunek 5: Twarze zredukowane $J=10$ (HOSVD)

Na ilustracji 6 przedstawiono wynik grupowania metodą k-średnich dla $J=20$ oraz liczby grup równej liczbie osób wynoszącej 8.



Rysunek 6: Twarze zredukowane $J=20$ (HOSVD)

Na rysunku 7 przedstawiono wynik grupowania metodą k-średnich dla $J=30$ oraz liczby grup równej liczbie osób wynoszącej 8.



Rysunek 7: Twarze zredukowane $J=30$ (HOSVD)

W tabeli 1 zamieszczono otrzymane wartości metryk. Dla algorytmu k-średnich zostały policzone Acc (dokładność) oraz Rand's index.

Tabela 1: Otrzymane metryki dla różnych wartości parametru J (k-średnich)

	4	10	20	30
Acc (dokładność)	76,25	100	80,00	63,75
Rand's index	92,56	100	93,04	84,56

Jak możemy zauważyć metoda k-średnich daje najlepsze rezultaty dla $J=10$, zarówno mniej jak i więcej szczegółów w obrazie negatywnie wpływa na rezultat grupowania. W przypadku metody najbliższych sąsiadów jest inaczej – tutaj im bardziej szczegółowy obraz otrzyma ta metoda na wejściu tym lepsze będą rezultaty. Trzeba również pamiętać, że metoda k-średnich nie wymaga zbioru treningowego i uczącego.

W tabeli 2 zamieszczono metryki dla klasyfikacji metodą najbliższych sąsiadów. Zostały

policzone dokładność (Acc) oraz Rand's index dla klasyfikacji w przestrzeni cech $\hat{U}^{(3)}$ przy pomocy metody najbliższych sąsiadów.

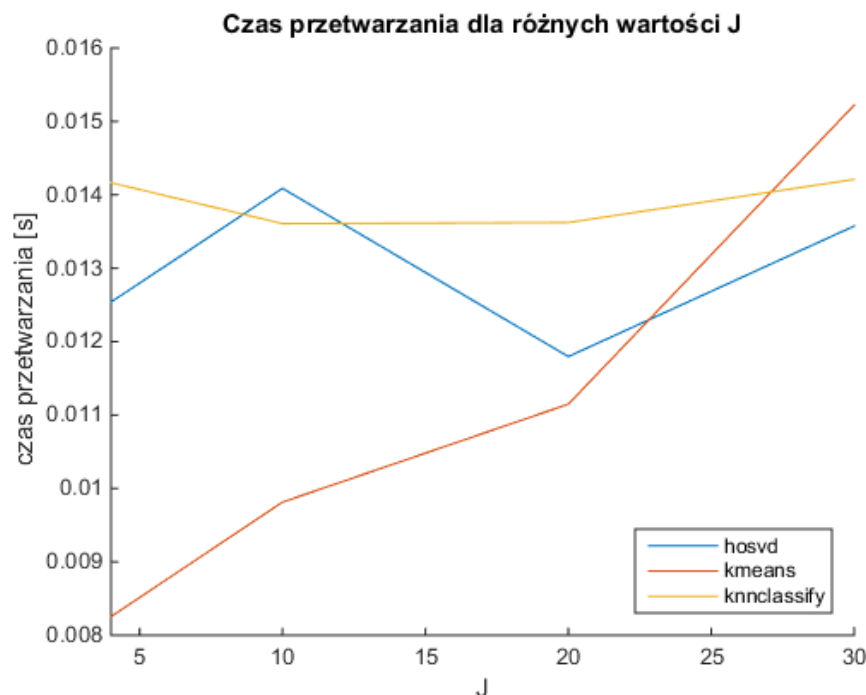
Tabela 2: Otrzymane metryki dla różnych wartości parametru J (k-najbliższych sąsiadów)

	4	10	20	30
Acc (dokładność)	81,25	93,75	100	100
Rand's index	94,17	97,50	100	100

Natomiast zależności czasowe przedstawiono w tabeli 3. Grupowanie przeprowadzono dla ilości grup równej liczbie osób aby móc jednoznacznie zinterpretować wyniki. Na ilustracji 8 zamieszczono graficzne porównanie.

Tabela 3: Czas przetwarzania [ms] w zależności od parametru J

	4	10	20	30
HOSVD	12,54	14,09	11,80	13,58
grupowanie k-średnich	8,25	9,81	11,15	15,22
klasyfikacja k-NN	14,17	13,61	13,62	14,21



Rysunek 8: Czas przetwarzania w zależności od wartości J

4 Podsumowanie

Podczas prac nad zadaniami zapoznano się z metodami redukcji wymiarowości przy pomocy nieujemnej faktoryzacji macierzy i dekompozycji tensorów.

Literatura

- [1] Dokumentacja środowiska MATLAB, <https://www.mathworks.com/>
- [2] Zdunek, Rafał, „Nieujemna faktoryzacja macierzy i tensorów : zastosowanie do klasyfikacji i przetwarzania sygnałów”, Oficyna Wydawnicza Politechniki Wrocławskiej, 2014
- [3] <http://www.sandia.gov/~tgkolda/TensorToolbox/index-2.6.html>
- [4] <http://www.esat.kuleuven.be/sista/tensorlab/>
- [5] <http://www.bsp.brain.riken.jp/TDALAB/>
- [6] <http://www.bsp.brain.riken.jp/~phan/>
- [7] <http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html>