Inteligencja obliczeniowa i jej zastosowania

Laboratorum cz. IV, nr 1-2

Autorzy:
Joanna Piątek, nr indeksu: 199966
Agnieszka Wątrucka, nr indeksu: 200016
Grupa: Środa, 15:15

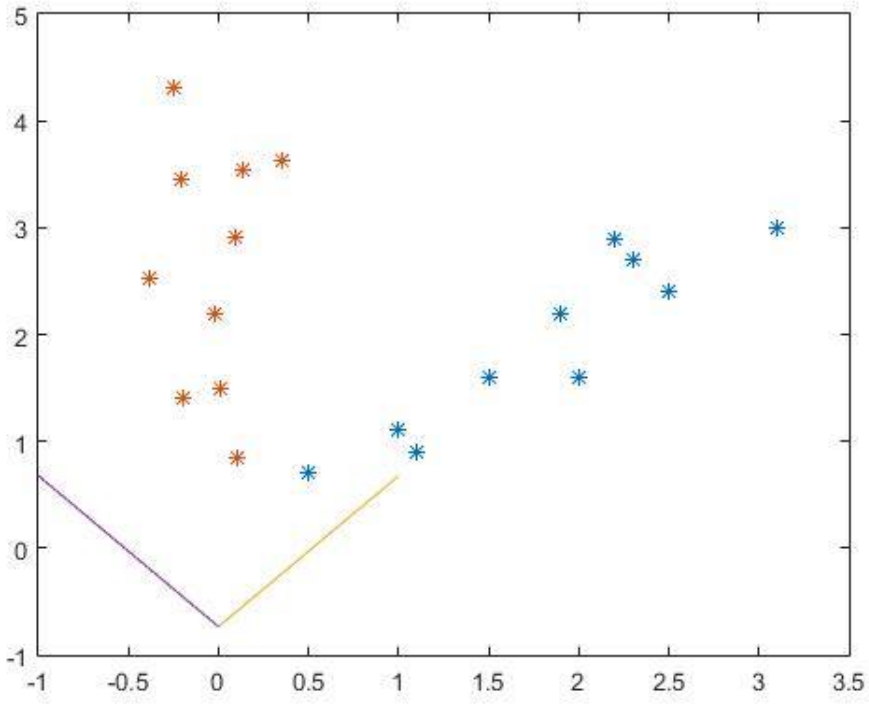
7 czerwca 2017

Prowadzący: prof. dr hab. inż. Rafał Zdunek

1 Zadanie nr 1

W ćwiczeniu nr 1 za zadanie była implementacja metody PCA w środowisku Matlab, wyznaczenie składowych głównych i wektora cech oraz wskazać na rysunku punkty obserwacji oraz wyznaczone wielkości.

1.1 Wyniki



Rysunek 1: Wykres zawierający wartości własne oraz wyznaczone wektory wartości własnych

Na wykresie zostały przedstawione punkty obserwacji i wektory wartości własnych wyznaczone za pomocą zaimplementowanej metody PCA.

1.2 Implementacja

```
X=[2.5 0.5 2.2 1.9 3.1 2.3 2 1 1.5 1.1; 2.4 0.7 2.9 2.2 3 2.7 1.6 1.1 1.6 0.9];

J = 2;

covX = cov(X');
[V, D] = eigs(covX,J);
Z = V'*X;

clf;
plot(X(1, :), (X(2, :)), '*');
hold on;
plot(Z(1, :), (Z(2, :)), '*');
hold on;
plot([0,1],V(1,:))
hold on;
plot([-1,0],[V(2,1), V(1,1)])
```

Rysunek 2: Implementacja algorytmu PCA i rysowanie wykresu dla zadanych danych

2 Zadanie 2

2.1 Implementacja główna

Na poniższym listingu została przedstawiona główna pętla programu. Podczas jej działania zostają wykonane następujące operacje:

- Wczytanie wybranej ilości folderów ze zdjęciami,
- Utworzenie wzorcowego wektora, zawierającego klasy kolejnych obrazów (klasa - numer osoby, której zrobiono zdjęcie),
- Dla każdej z wskazanych wartości estymowanych komponentów głównych (J) zostaje przeprowadzone zredukowanie wymiarów za pomocą funkcji *myPCA*,
- Zdjęcia oryginalne oraz zredukowane zostają poddane grupowaniu i klasyfikacji,
- Zapis skuteczności i czasu działania algorytmów do plików.

Pętla ta jest wykonywana dla różnej liczby klas (grup) zdjęć, w zależności od zmiennej *classes_count*.

```
classes_count = 5;
% Wczytywanie zdjec dla wybranej liczy klas
images_arrays = getAllImages(classes_count);

J = [4, 10, 20, 30];
iterations = 10;

% Wektor zawierajacy klasy kolejnych obrazow
```

```
images_classes = [];
for i = 1:classes_count
        images_classes =[images_classes; ones(10,1).*i];
end

% Petla glowna
for j_val = 1:4
        grouping_results = zeros([iterations 4]);
        classification_results = zeros([iterations 4]);

        for i = 1 : iterations
                % Redukcja wymiarow
                [V, pca_images_arrays, D] = myPCA(images_arrays', J(j_val));

                % Grupowanie za pomoca k-srednich
                g_result = getGroupingResults(
                images_arrays, pca_images_arrays, classes_count,
                images_classes);
                grouping_results(i,:) = g_result;

                % Klasyfikacja z uzyciem k-NN
                c_result = getClassificationResults(
                images_arrays, pca_images_arrays, classes_count,
                images_classes);
                classification_results(i,:) = c_result;
        end

        % Zapis wynikow do plikow
        (...)
end
```

W celu redukcji wymiarów za pomocą algorytmu PCA, została zaimplementowana funkcja *myPCA*, którą przedstawiono poniżej.

```
function [V, newX, D] = myPCA(X, J)
        X = bsxfun(@minus, X, mean(X,2));
        C = (X*X')./(size(X,2)-1);

        [V, D] = eigs(C,J);
        [D, order] = sort(diag(D), 'descend');
        V = V(:, order);
        newX = V'*X;
end
```

2.2 Implementacja grupowania

Pierwszym krokiem do rozwiązania zadania było pogrupowanie zdjęć z użyciem algorytmu k-średnich. W tym celu powstała funkcja *getGroupingResults*.

```
function result = getGroupingResults(images_arrays, pca_images_arrays,
        clusters, images_classes)

        % Uruchomienie grupowania k-srednich,
        % Zapis wynikow i czasu trwania
        tic;
        groups = kmeans(images_arrays, clusters);
        default_time = toc;
        tic;
        groups_pca = kmeans(pca_images_arrays', clusters);
        pca_time = toc;

        % Obliczenie skutecznosci grupowania i jej normalizacja
        default_acc = AccMeasure(images_classes, groups)/100.0;
        pca_acc = AccMeasure(images_classes, groups_pca)/100.0;

        result = [default_acc, pca_acc, default_time, pca_time];
end
```

Jak można zauważyć na powyższym listingu, funkcja przyjmuje 4 argumenty:

- *images_arrays* - oryginalne macierze zdjęć,
- *pca_images_arrays* - macierze zdjęc po redukcji wymiarów,
- *clusters* - ilość klastrów (grup) zdjęć,
- *images_classes* - wzorzec klas dla kolejnych zdjęć.

Podczas wykonywania funkcji mierzony jest czas grupowania, a następnie, za pomocą funkcji *AccMeasure*, także jej skuteczność. Numery klas przyporządkowanych kolejnym obrazom w wyniku grupowania k-średnich nie zawsze są tożsame z tymi, które zostały podane jako wzorcowe. Jednak *AccMeasure* radzi sobie z przyporządkowaniem klas założonych klasom otrzymanym ze skutecznością ponad 80%. Można to sprawdzić, uruchamiając ją w ten sposób, by zwracała trzy wartości - jedna z nich przedstawia właśnie tą skuteczność. Wyniki otrzymane za pomocą tej funkcji należy znormalizować tak, by największa możliwa ich wartość była równa 1. Wtedy można łatwo porównać je z wynikami klasyfikacji.
Ostatecznie otrzymujemy skuteczność i czas grupowań zdjęć oryginalnych i tych po redukcji wymiarów.

2.3 Implementacja klasyfikacji

Kolejnym elementem implementacji jest klasyfikacja obrazów za pomocą algorytmu k najbliższych sąsiadów. W tym celu została stworzona funkcja *getClassificationResults*. Przyj-

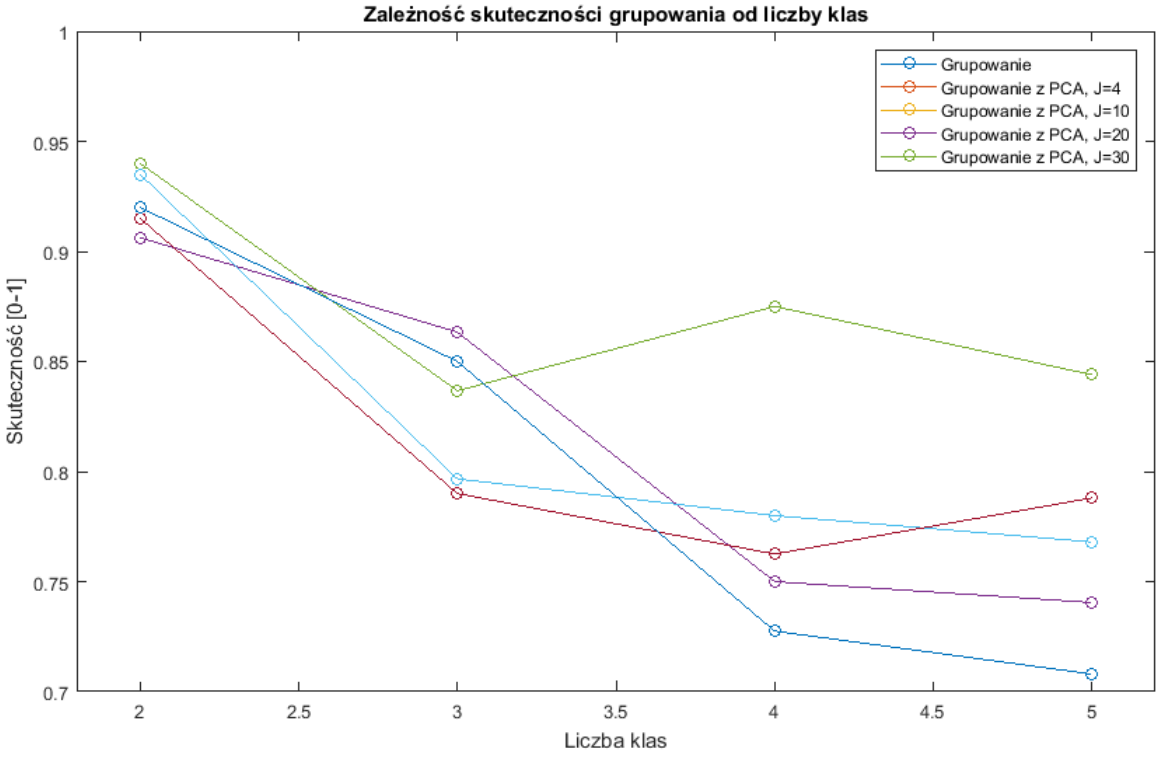
muje ona te same argumenty, co funkcja używana przy grupowaniu, jednak dla lepszego zrozumienia działania kodu zmiennej *clusters* z poprzedniego listingu odpowiada *classes_count* z obecnego.

```
function result = getClassificationResults(images_arrays, pca_images_arrays,
 classes_count, images_classes)

        % Uruchomienie klasyfikacji k-NN,
        % Zapis wynikow i czasu trwania
        tic;
        model = fitcknn(images_arrays, images_classes);
        cv_model = crossval(model,'KFold',classes_count);
        default_time = toc;
        tic;
        model_pca = fitcknn(pca_images_arrays', images_classes);
        cv_model_pca = crossval(model_pca,'KFold',classes_count);
        pca_time = toc;

        % Obliczenie skutecznosci grupowania
        cv_model_loss = kfoldLoss(cv_model);
        default_acc = 1 - cv_model_loss;
        cv_model_pca_loss = kfoldLoss(cv_model_pca);
        pca_acc = 1 - cv_model_pca_loss;

        result = [default_acc, pca_acc, default_time, pca_time];
```

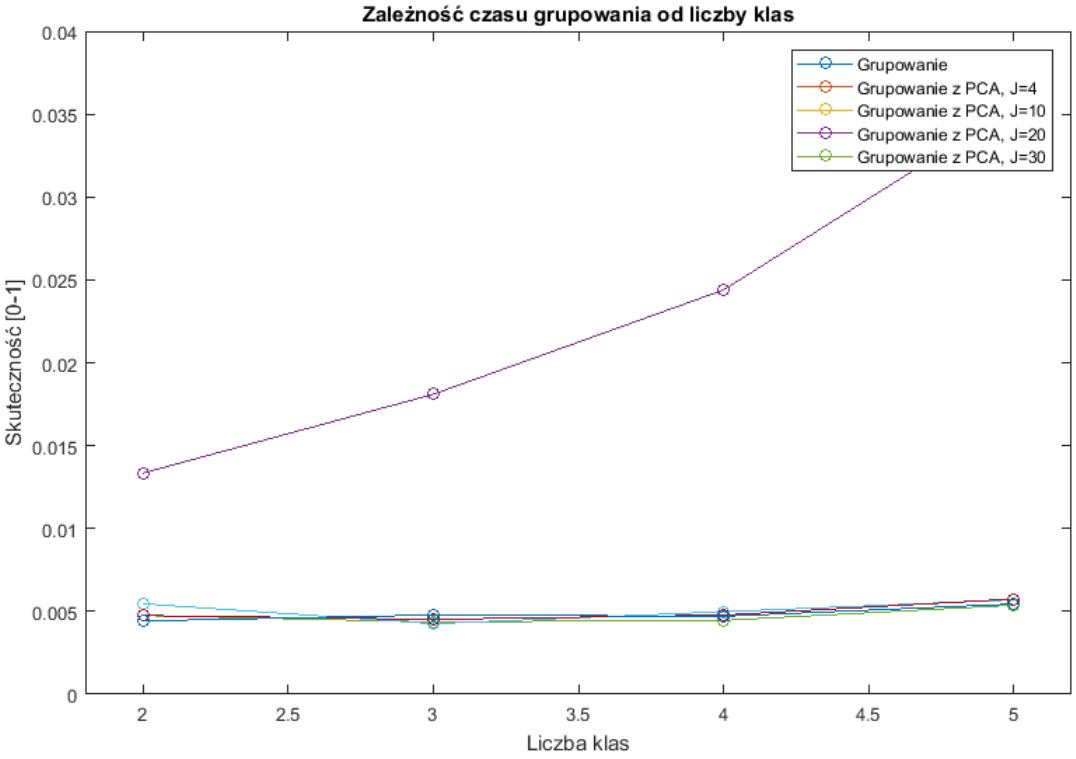
Najpierw, na podstawie macierzy obrazów i wzorca ich klas, za pomocą algorytmu k-NN, tworzony jest model klasyfikatora. Następnie, z wykorzystanie fukcji *crossval*, przeprowadzona zostaje kroswalidacja, czyli analiza modelu, polegająca na wybraniu przez algorytm różnych elementów zbioru jako zestaw uczący, a jednego jako testowy. Taki sprawdzian zostaje przeprowadzony dla wielu kombinacji elementów w grupach. Ostatnim krokiem jest sprawdzenie błędu modelu za pomocą funkcji *kfoldLoss*. Odejmując wynik od liczby 1 otrzymujemy skuteczność klasyfikacji w modelu.

2.4 Wyniki

2.4.1 Grupowanie

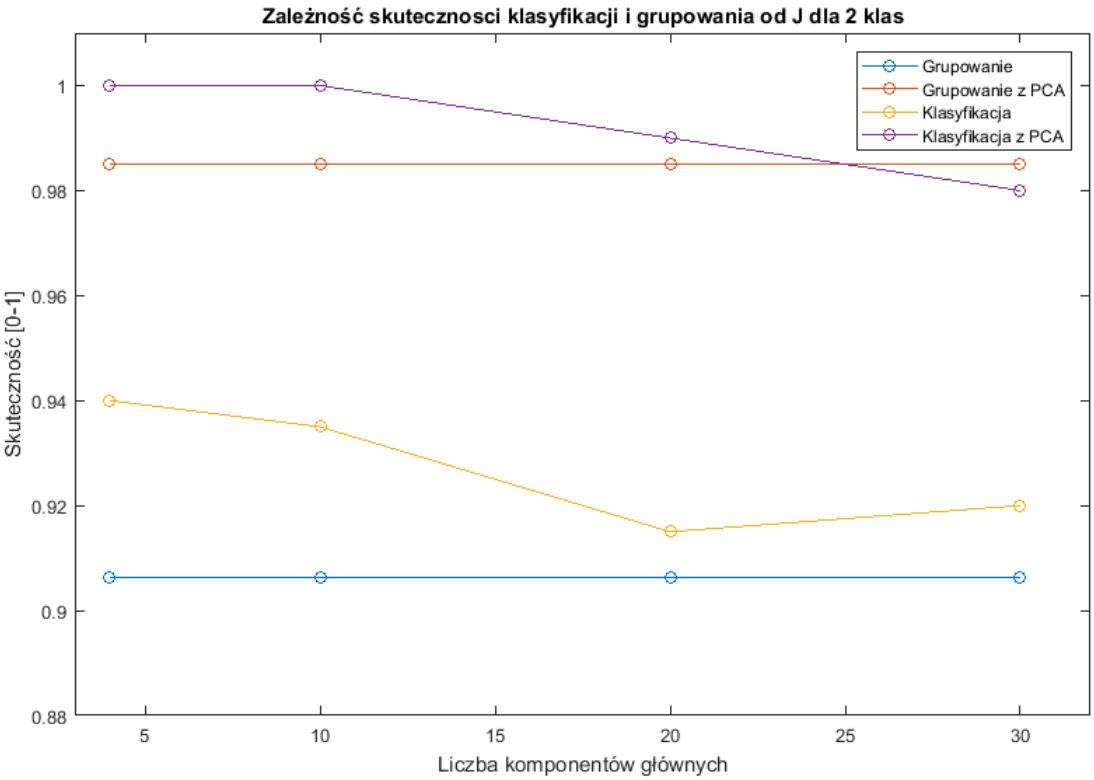


Rysunek 3: Skuteczność grupowania dla wymiarów pełnych i zredukowanych

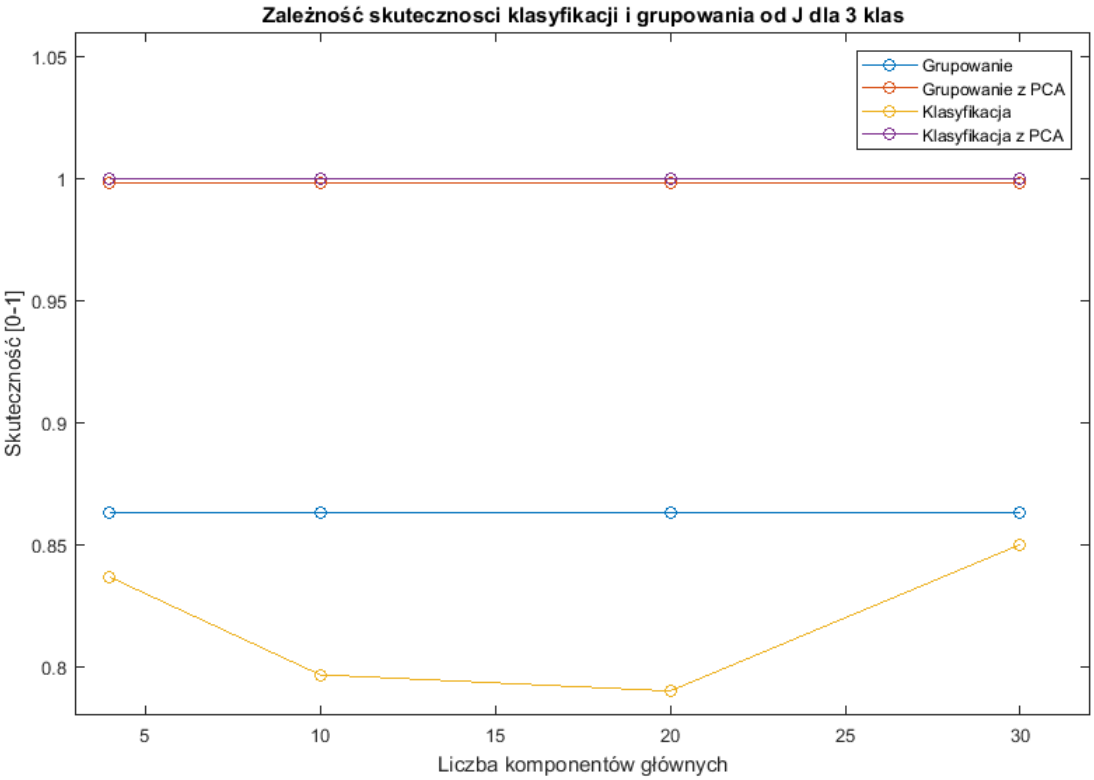


Rysunek 4: Czas grupowania dla wymiarów pełnych i zredukowanych

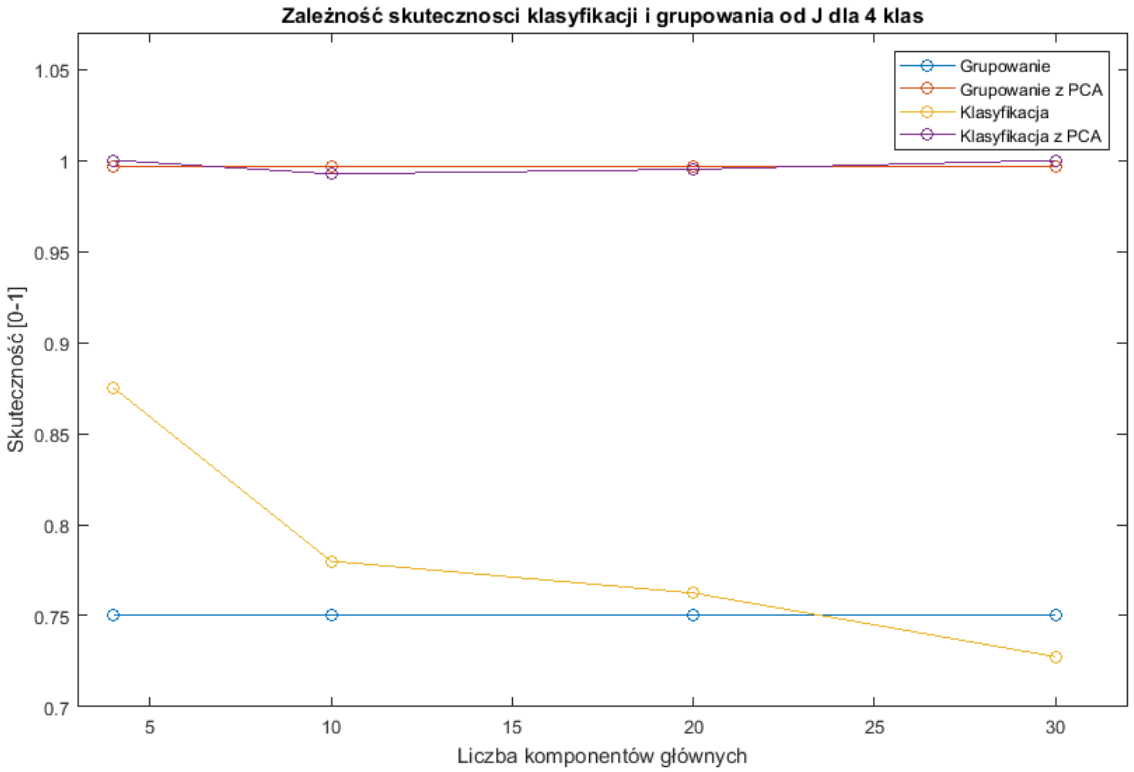
2.4.2 Porównanie klasyfikacji i grupowania



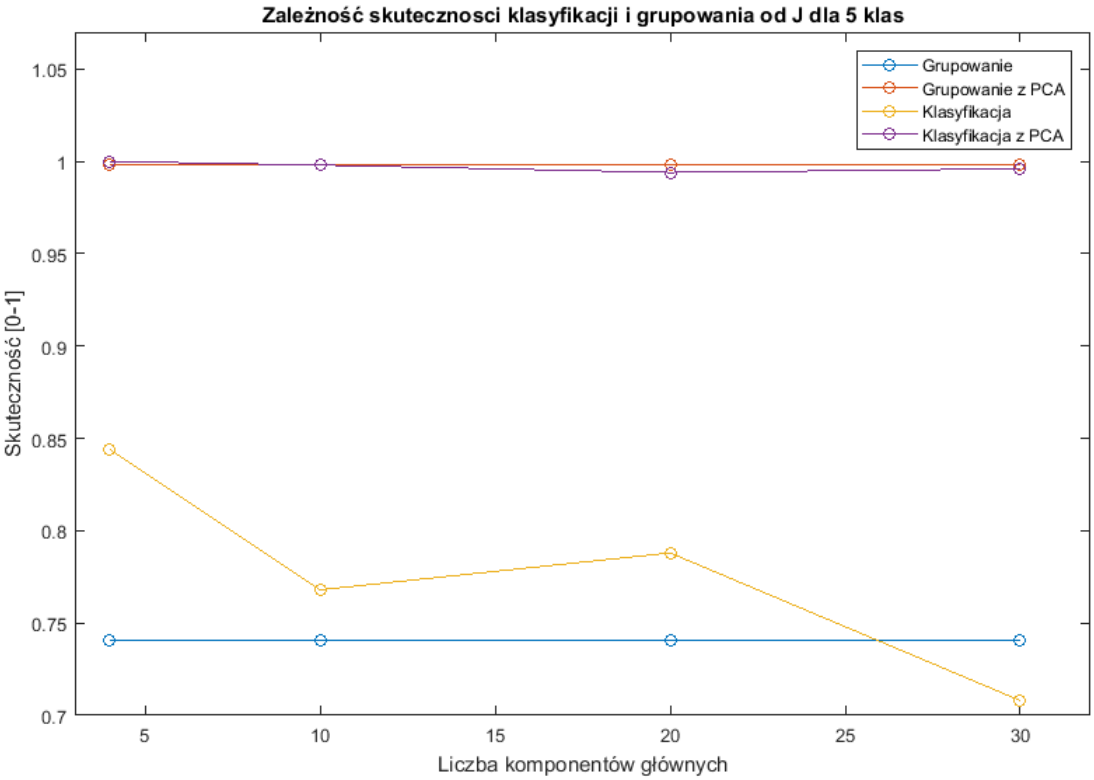
Rysunek 5: Skuteczność grupowania i klasyfikacji dla 2 klas



Rysunek 6: Skuteczność grupowania i klasyfikacji dla 3 klas



Rysunek 7: Skuteczność grupowania i klasyfikacji dla 4 klas



Rysunek 8: Skuteczność grupowania i klasyfikacji dla 5 klas