Krzysztof Barden 210139 210139@edu.p.lodz.pl
Adam Troszczyński 210342 210342@edu.p.lodz.pl

Zadanie 1.: Sieci neuronowe

1. Cel

Zadanie polega na implementacji algorytmów umożliwiające znalezienie najlepszego rozkładu neuronów samoorganizującej się sieci neuronowej, który to rozkład jak najlepiej odzwierciedla rozkład zadanych punktów z tego zbioru
Użyte przez nas algorytmy to:
- Algorytm k-średnich
- Algorytm Kohonena.

2. Wprowadzenie

Algorytm k-średnich to iteracyjny algorytm, w którym początkowo w przestrzeni z danymi rozmieszczane jest k centrów(centroidów). Następnie w każdej iteracji do każdego z centrów przyporządkowywane są dane wejściowe na podstawie ich odległości do centrum oraz aktualizowane jest położenie centrum, które wyznaczane jest jako średnia z przyporządkowanych do niego danych. Iteracje te powtarzane są tak długo aż położenie centrów ulegnie stabilizacji.

Sieć realizująca klasyczną samoorganizującą się mapę składa się ze zbioru neuronów. Neurony rywalizują za sobą o prawo do reprezentowania wzorca wejściowego, oznacza to, że ten który odpowie najsilniej jest ostatecznie przyjmowany jako reprezentant wzorca. Funkcja transmisji neuronów jest przeważnie funkcją odległości między wektorem wejściowym a wektorem wag, a dany neuron możemy zinterpretować jako punkt/wektor w przestrzeni wzorców wejściowych. Znalezienie optymalnego rozkładu neuronów w przestrzeni możliwe jest za pomocą algorytmu Kohonena.

Algorytm Kohonena - Klasyczny algorytm Kohonena zakłada adaptację wag jedynie neuronu zwycięskiego i neuronów znajdujących się nie dalej niż

obliczony promień sąsiedztwa.
Odległość od neuronu jest odległoscią Euklidesową wyrażoną wzorem:

$$Distance = \sqrt{\sum_{i=0}^{i=n}(V_i - W_i)^2} \tag{1}$$

Gdzie:
V - obecny zadany wektor
W - wektor wag zadanego węzła
i - numer iteracji
Po wyłonieniu zwycięzcy (Best Matching Unit - BMU) , należy obliczyć które inne węzły są w sąsiedztwie BMU. Wszystkie te węzły będą miały poprawioną wagę w kolejnym kroku. W algorytmie Kohonena odległość sąsiedztwa maleje z czasem zgodnie z wzorem:

$$\sigma(t) = \sigma_0 exp(-\frac{t}{\lambda}) \tag{2}$$

Gdzie:
t =1,2,3 ... (numer iteracji)
lambda - wybrana stała czasowa
Każdy węzeł w sąsiedztwie BMU (włacznie z BMU) ma poprawianą wagę zgodnie ze wzorem:

$$W(t+1) = W(t) + \theta(t)L(t)(V(t) - W(t) \tag{3}$$

Gdzie:
L(t) - współczynnik nauki (learning rate)

$$L(t) = L_0 exp(-\frac{t}{\lambda}) \tag{4}$$

theta(t) - wpływ BMU na jego learning rate

$$\theta(t) = exp(-\frac{Distance^2}{2\sigma^2(t)}) \tag{5}$$

3. Opis implementacji

Do wykonania zadania został użyty język Python

Algorytm k-srednich:
Algorytm używa struktur danych :
Centroid przechowujący swoją pozycję,
Data przechowujący wektory,
Cluster przechowujący swoje wektory i przypisany centroid
Po wczytaniu danych centroidy zostają losowo rozmieszczone. Następnie w

każdej iteracji wektory zostają przypisane do najbliższego z centroidów, centroidy zostają przesunięte i sprawdzone czy nie mają zbyt mało wektorów. Algorytm kończy się gdy osiągnięto maksymalną ilosć iteracji lub przekroczono zadaną wartosć błędu kwantyzacji.

Algorytm Kohonena:
Algorytm używa struktur danych :
Neuron przechowujący swoją pozycję i wagę
Data przechowujący wektory,
Cluster przechowujący swoje wektory i przypisany centroid
Po wczytaniu danych neurony zostają losowo rozmieszczone. W każdej iteracji zostają obliczone BMU's, wagi oraz współczynnik nauki. Algorytm kończy się gdy osiągnięto maksymalną ilosć iteracji lub przekroczono zadaną wartosć błędu kwantyzacji.

4. Materiały i metody

Eksperyment nr 1 - Seeds data set
W tym eksperymencie algorytmy przetworzą Seeds data set.
W Każdym podpunkcie program będzie uruchamiany dwa razy
Dane wejsciowe dla algorytmów:
Podpunkt 1.1:
Algorytm k-srednich: liczba centroidów - 3, max liczba iteracji - 64, max błąd - 0.000001
Algorytm Kohonena: liczba neuronów - 3, max liczba iteracji - 64, max błąd - 0.0001 lambda - 20, początkowy współczynnik nauki - 0.5, początkowy promień - 1.5
Podpunkt 1.2:
K-srednich - zwiekszona tolerancja bledu
Kohonen - zwiększona lambda
Algorytm k-srednich: liczba centroidów - 3, max liczba iteracji - 64, max błąd - 0,.01
Algorytm Kohonena: liczba neuronów - 3, max liczba iteracji - 64, max błąd - 0.0001, lambda - 50, początkowy współczynnik nauki - 0.5, początkowy promień - 1.5
Podpunkt 1.3:
Kohonen - zmniejszony początkowy wspołczynnik nauki Nie ma sensu zmieniać innych danych w algorytmie k-srednich dlatego bedzie pomijany Algorytm Kohonena: liczba neuronów - 3, max liczba iteracji - 64, max błąd - 0.0001, lambda - 20, początkowy współczynnik nauki - 0.01, początkowy promień - 1.5
Podpunkt 1.4:
Kohonen- zwiększony początkowy promień sąsiedztwa
Algorytm Kohonena: liczba neuronów - 3, max liczba iteracji - 64, max błąd - 0.0001, lambda - 20, początkowy współczynnik nauki - 0.5, początkowy promień - 5
Podpunkt 1.5:

Kohonen - zwiększona tolerancja błędu
Algorytm Kohonena: liczba neuronów - 3, max liczba iteracji - 64, max błąd - 0.01, lambda - 20, początkowy współczynnik nauki - 0.01, początkowy promień - 1.5

Eksperyment nr 2 - Iris data set

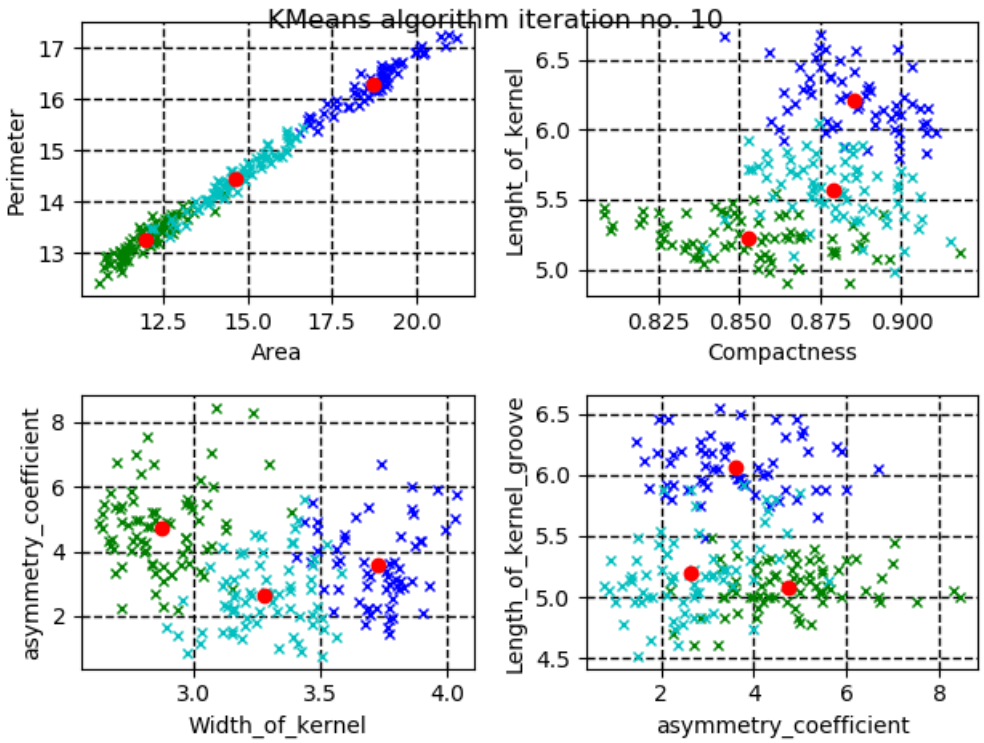
W tym eksperymencie algorytmy przetworzą Iris data set.
W Każdym podpunkcie program będzie uruchamiany dwa razy
Dane wejsciowe dla algorytmów:
Podpunkt 2.1:
Algorytm k-srednich: liczba centroidów - 3, max liczba iteracji - 64, max błąd - 0.000001
Algorytm Kohonena: liczba neuronów - 3, max liczba iteracji - 64, max błąd - 0.0001 lambda - 20, początkowy współczynnik nauki - 0.5, początkowy promień - 1.5
Podpunkt 2.2:
zwiększona tolerancja błędu
Algorytm k-srednich: liczba centroidów - 3, max liczba iteracji - 64, max błąd - 0.01
Algorytm Kohonena: liczba neuronów - 3, max liczba iteracji - 64, max błąd - 0.0001, lambda - 50, początkowy współczynnik nauki - 0.5, początkowy promień - 1.5

Eksperyment nr 2 - Abalone data set

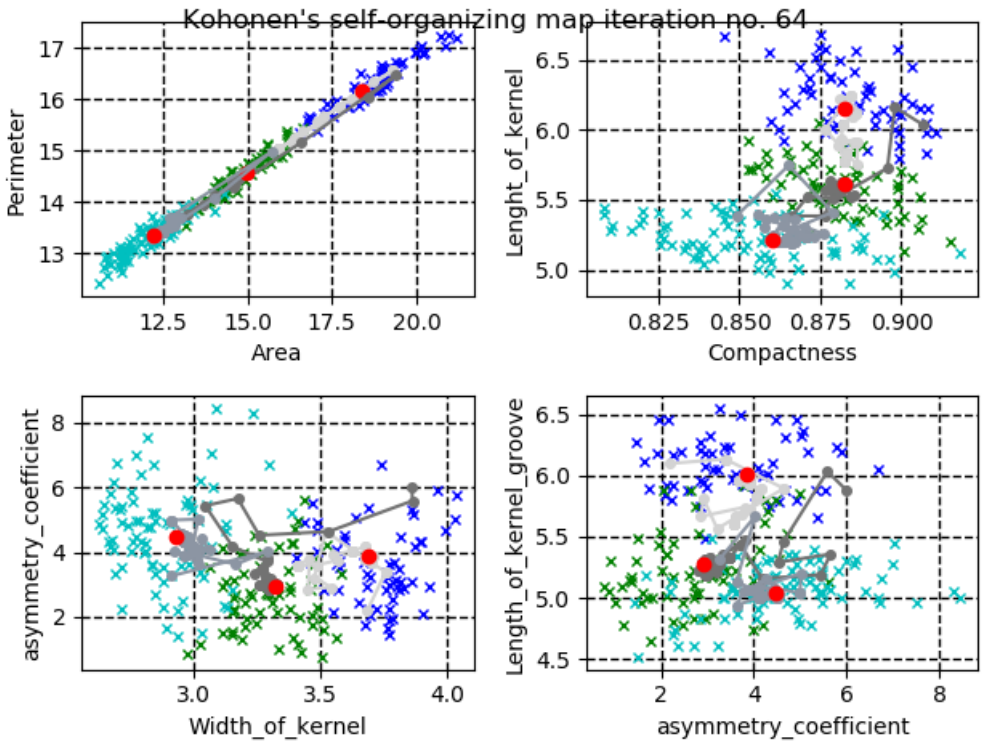
W tym eksperymencie algorytmy przetworzą Abalone data set o dużej ilosci danych
W Każdym podpunkcie program będzie uruchamiany dwa razy
Dane wejsciowe dla algorytmów:
Podpunkt 3.1:
Algorytm k-srednich: liczba centroidów - 3, max liczba iteracji - 64, max błąd - 0.000001
Algorytm Kohonena: liczba neuronów - 3, max liczba iteracji - 64, max błąd - 0.0001 lambda - 20, początkowy współczynnik nauki - 0.5, początkowy promień - 1.5
Podpunkt 3.2:
zwiększona tolerancja błędu
Algorytm k-srednich: liczba centroidów - 3, max liczba iteracji - 64, max błąd - 0.01
Algorytm Kohonena: liczba neuronów - 3, max liczba iteracji - 64, max błąd - 0.0001, lambda - 50, początkowy współczynnik nauki - 0.5, początkowy promień - 1.5

5. Wyniki

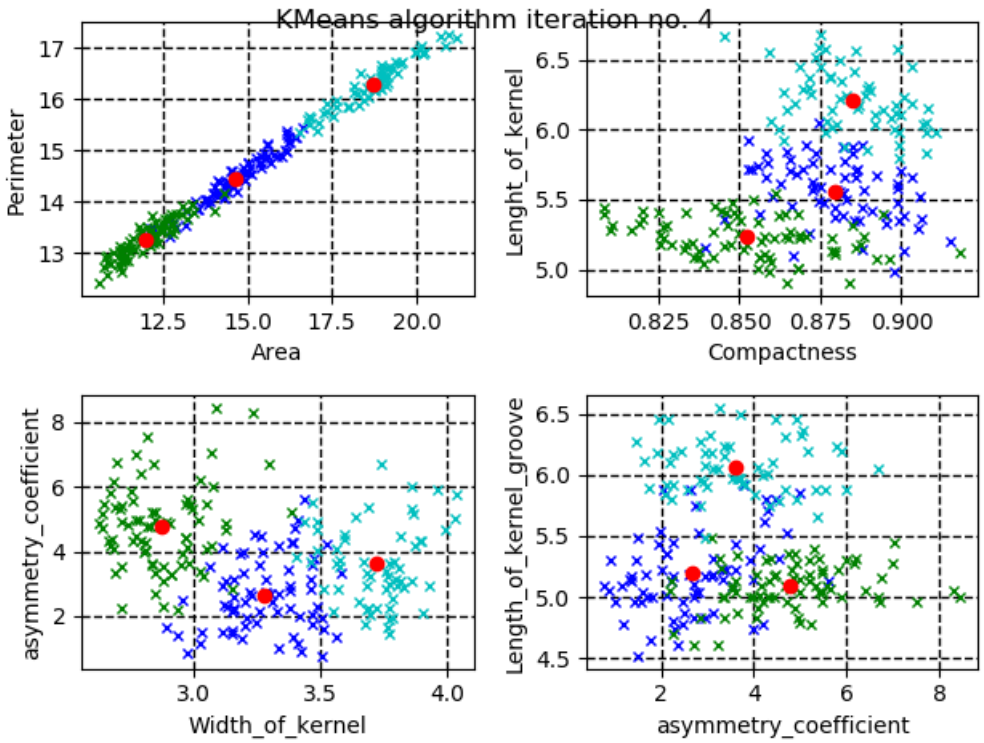
Podpunkt 1.1
Rysunek 1



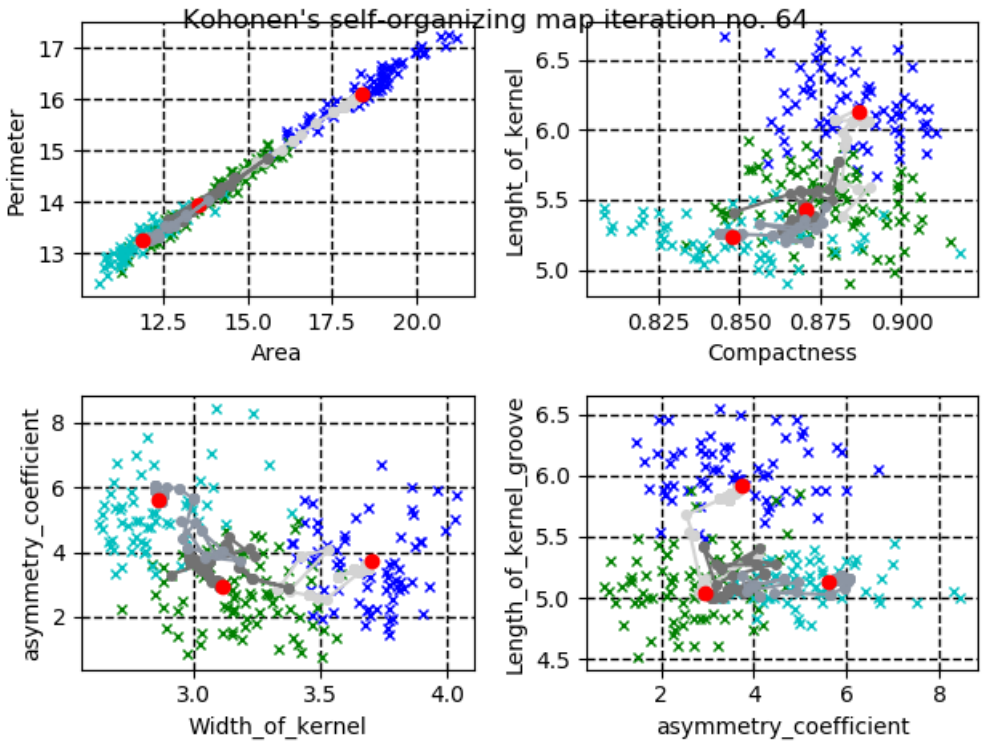
Rysunek 2



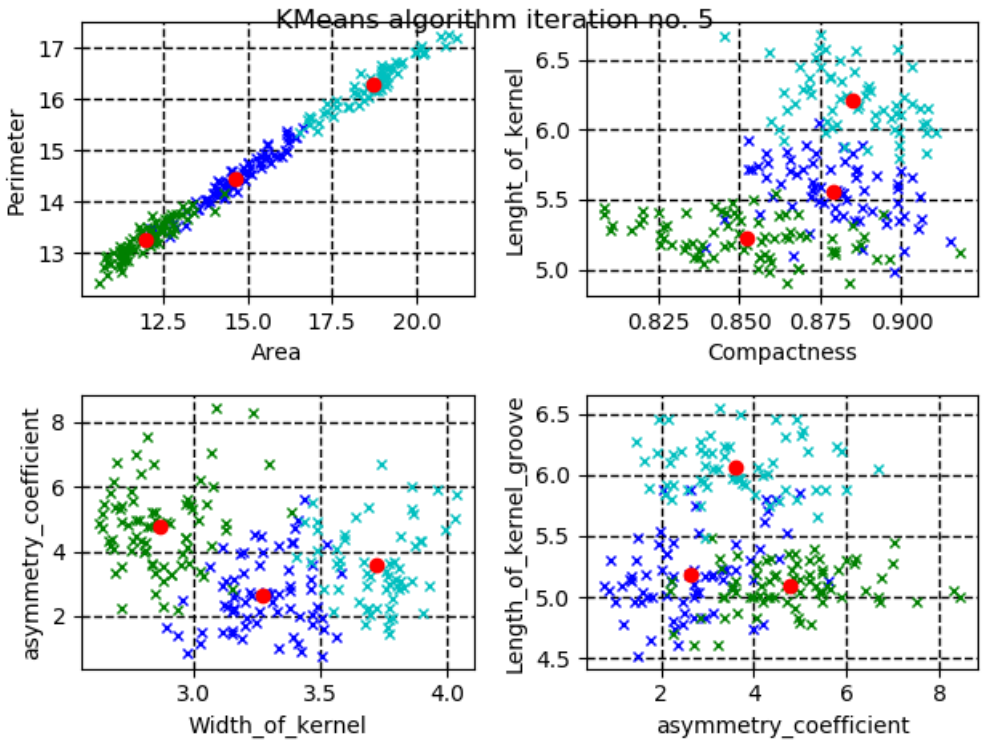
Rysunek 3



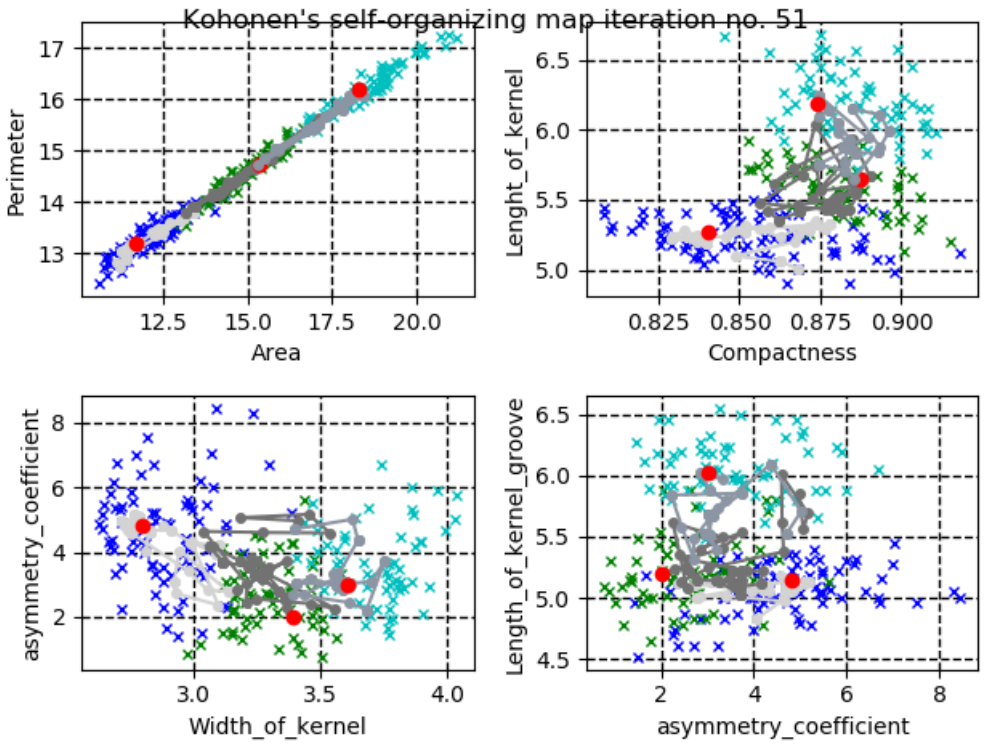
Rysunek 4



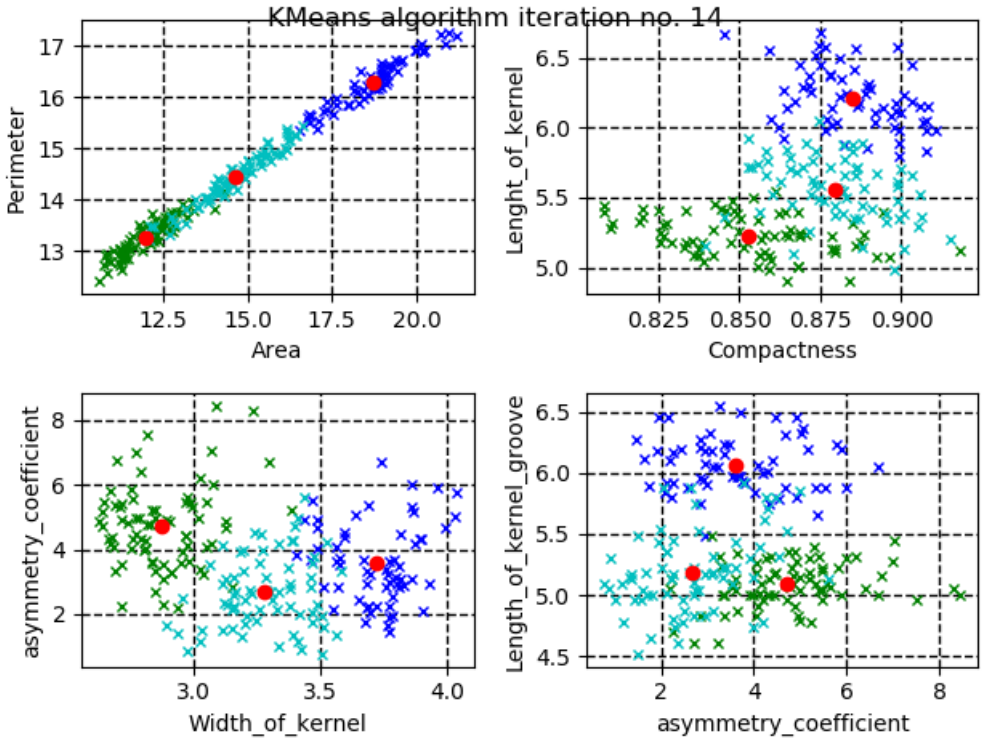
Podpunkt 1.2
K-srednich - zwiekszona tolerancja bledu
Kohonen - zwiększona lambda
Rysunek 5



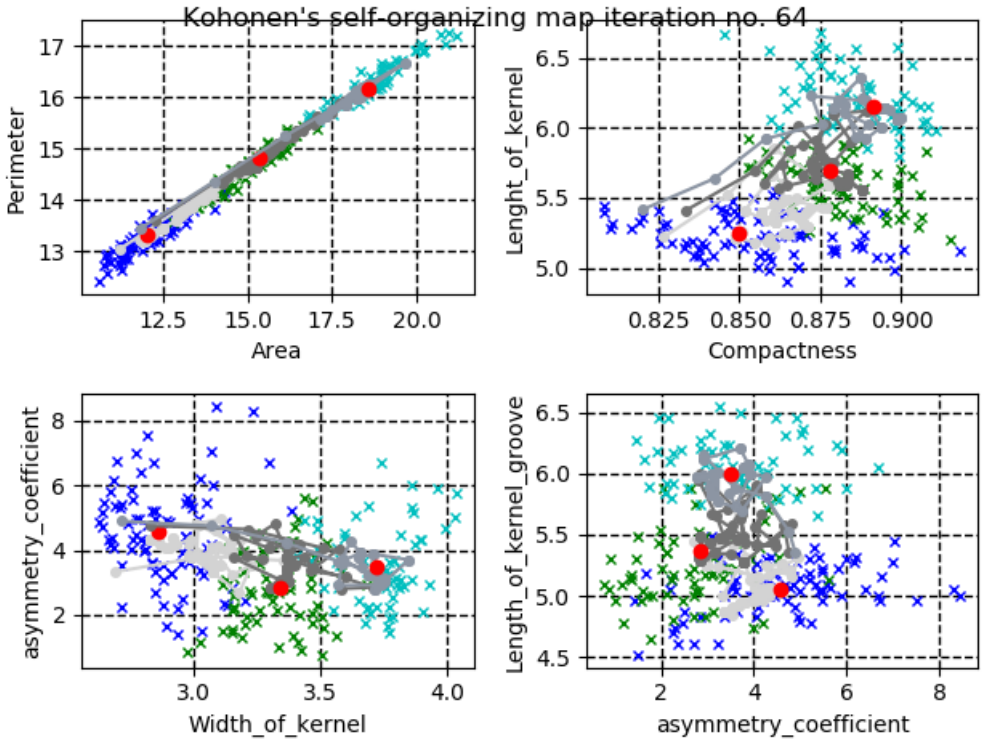
Rysunek 6



Rysunek 7

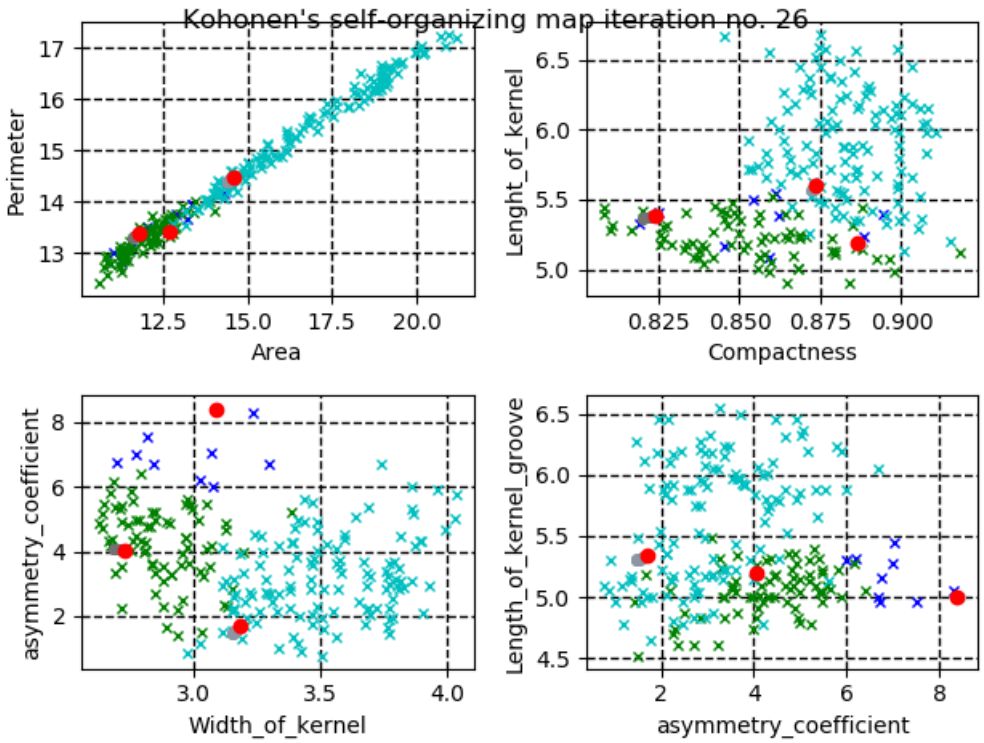


Rysunek 8

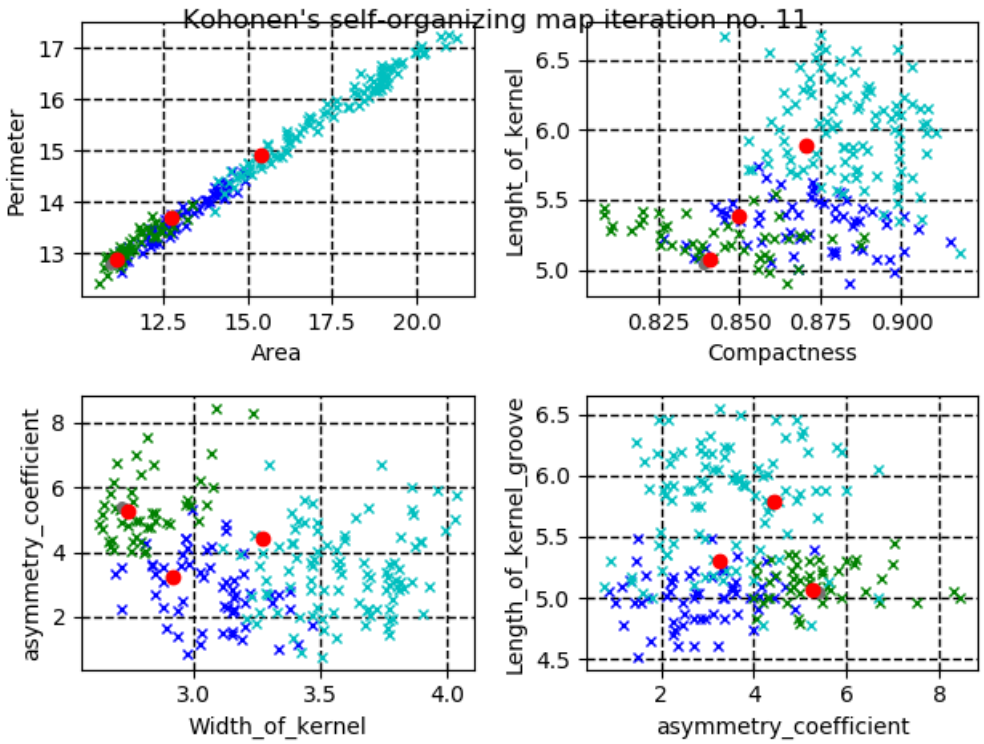


Podpunkt 1.3
Kohonen - zmniejszony początkowy współczynnik nauki

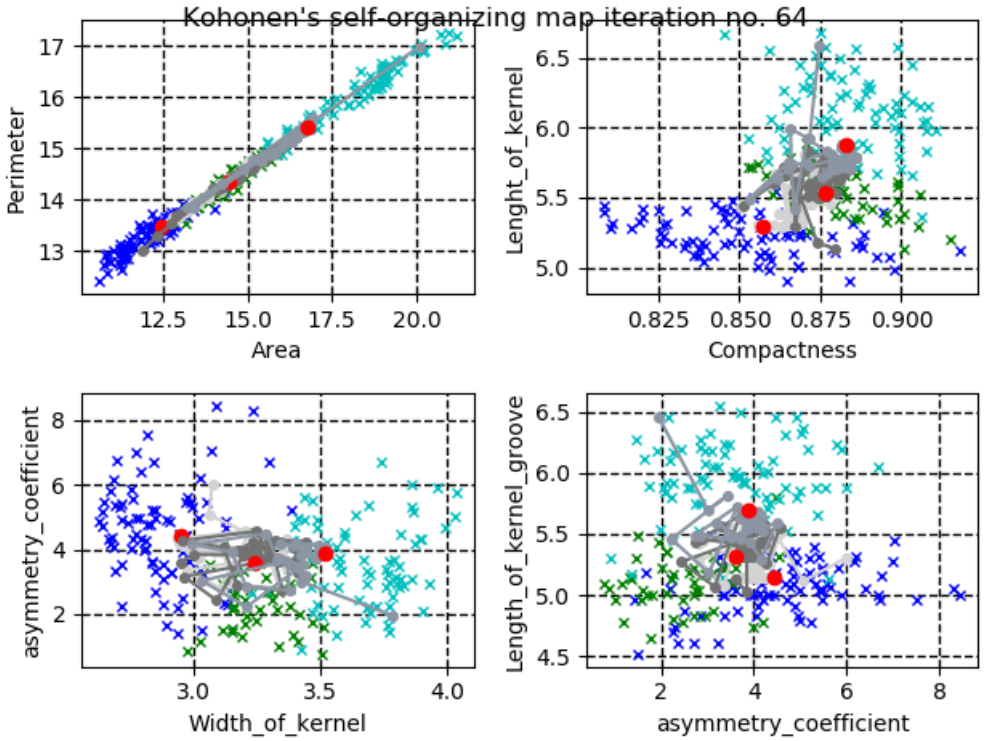
Rysunek 9



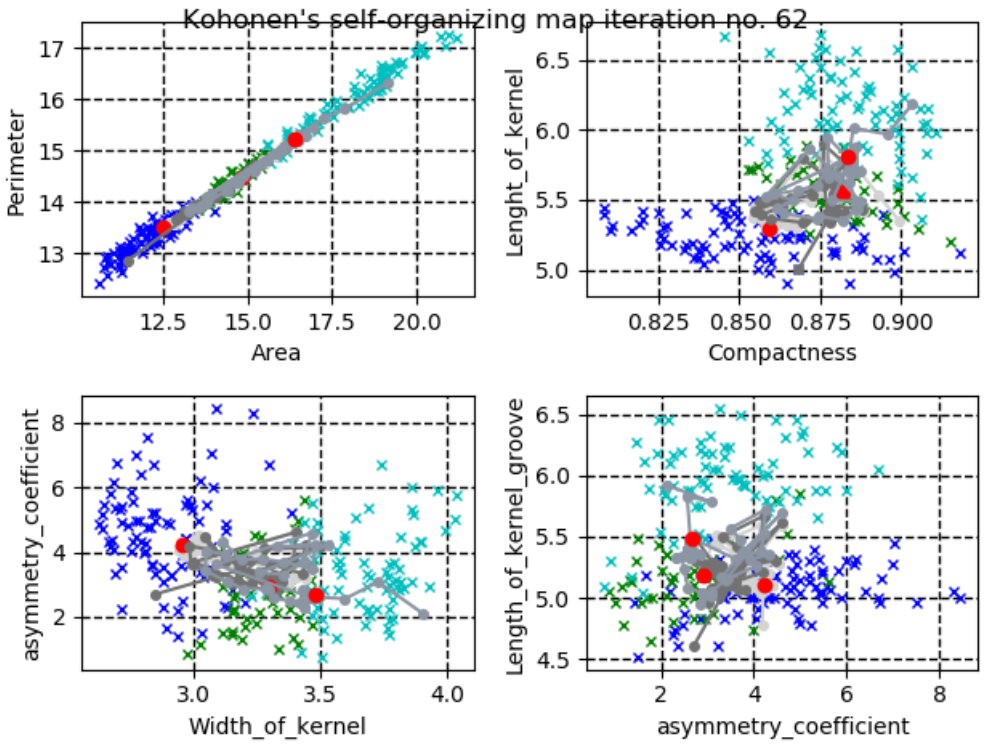
Rysunek 10



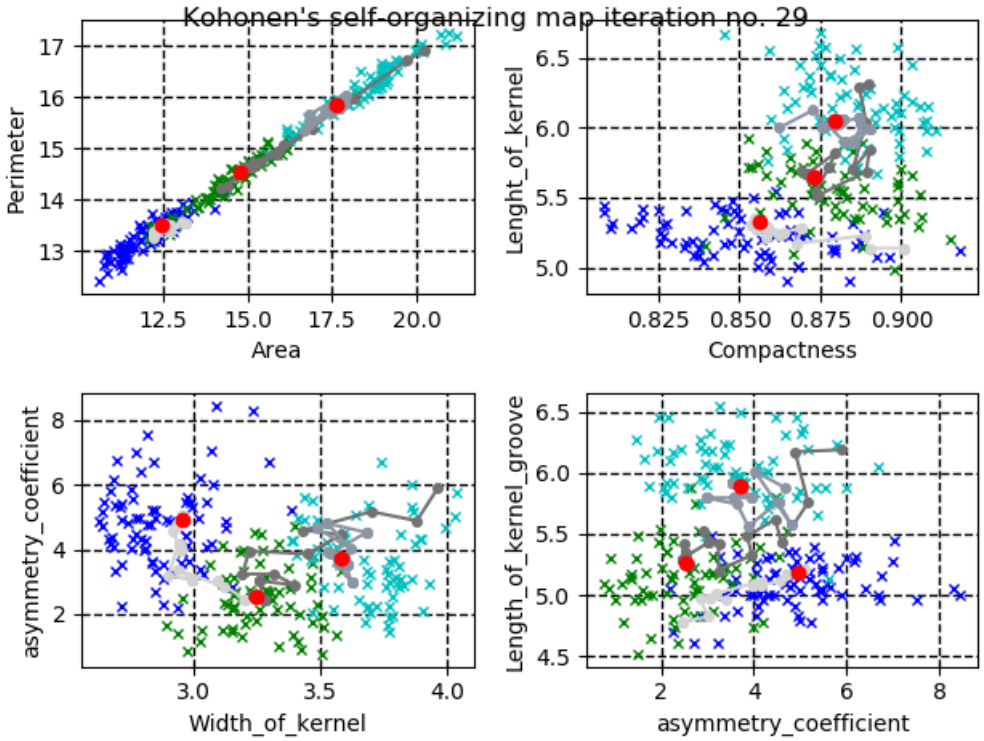
Podpunkt 1.4
Kohonen- zwiększony początkowy promień sąsiedztwa
Rysunek 11



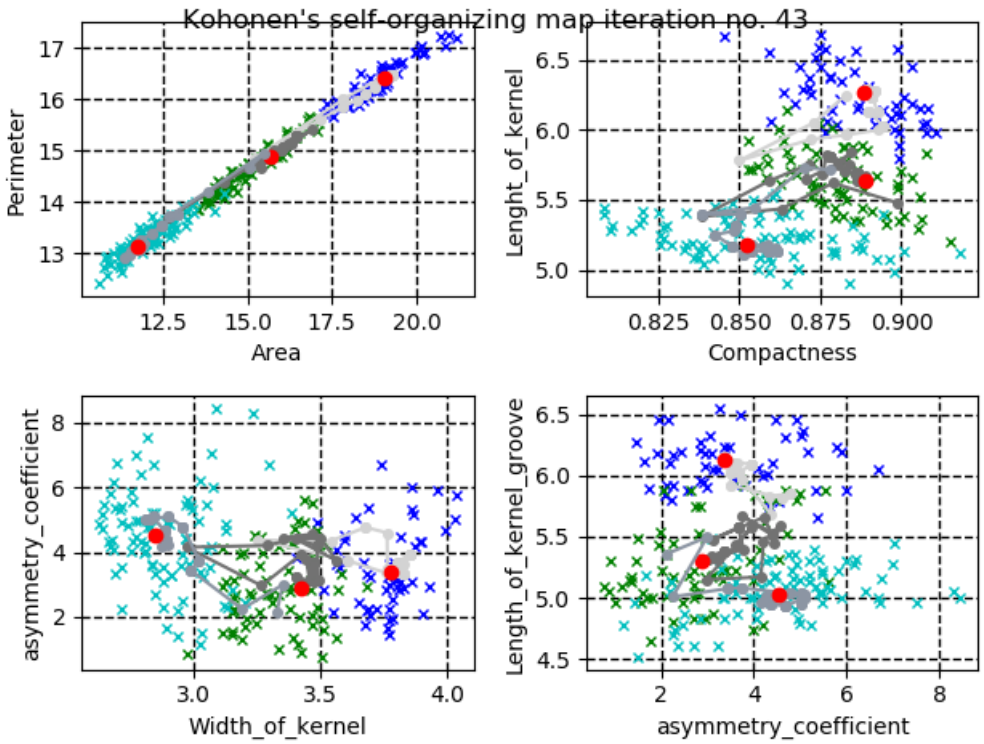
Rysunek 12



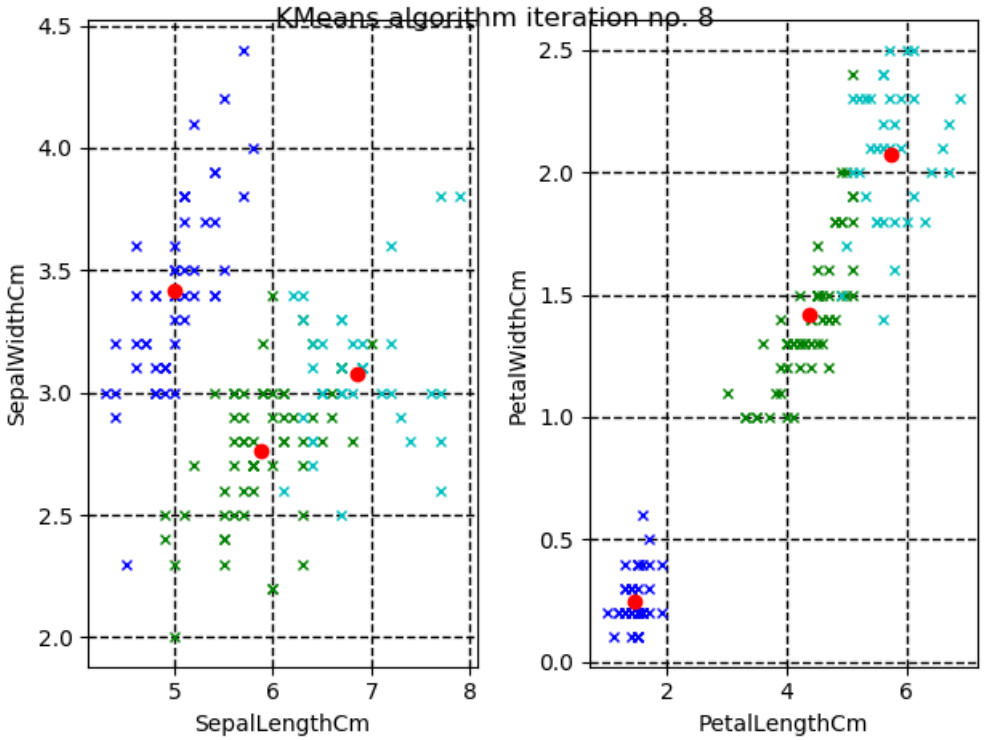
Podpunkt 1.5
Kohonen - zwiększona tolerancja błędu
Rysunek 13



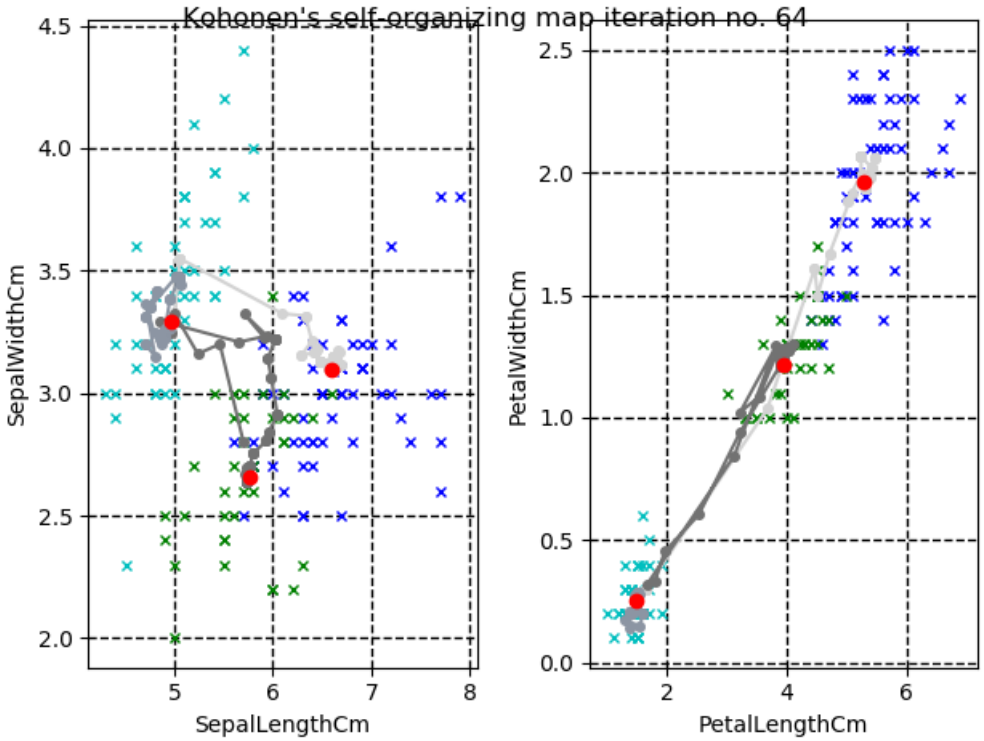
Rysunek 14



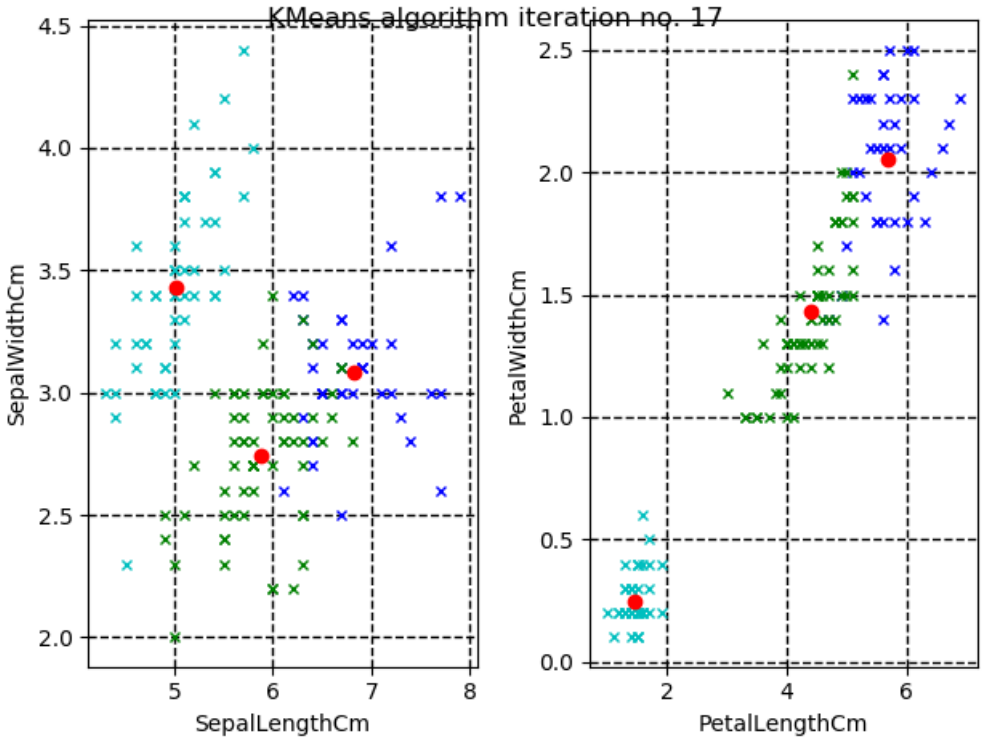
Podpunkt 2.1
Rysunek 15



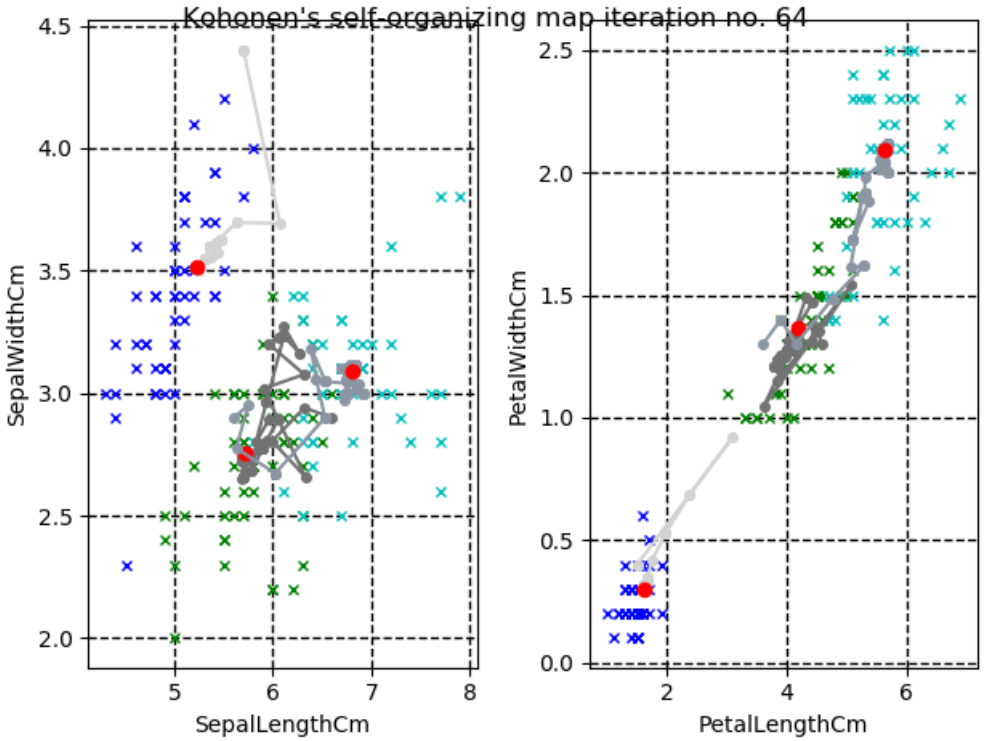
Rysunek 16



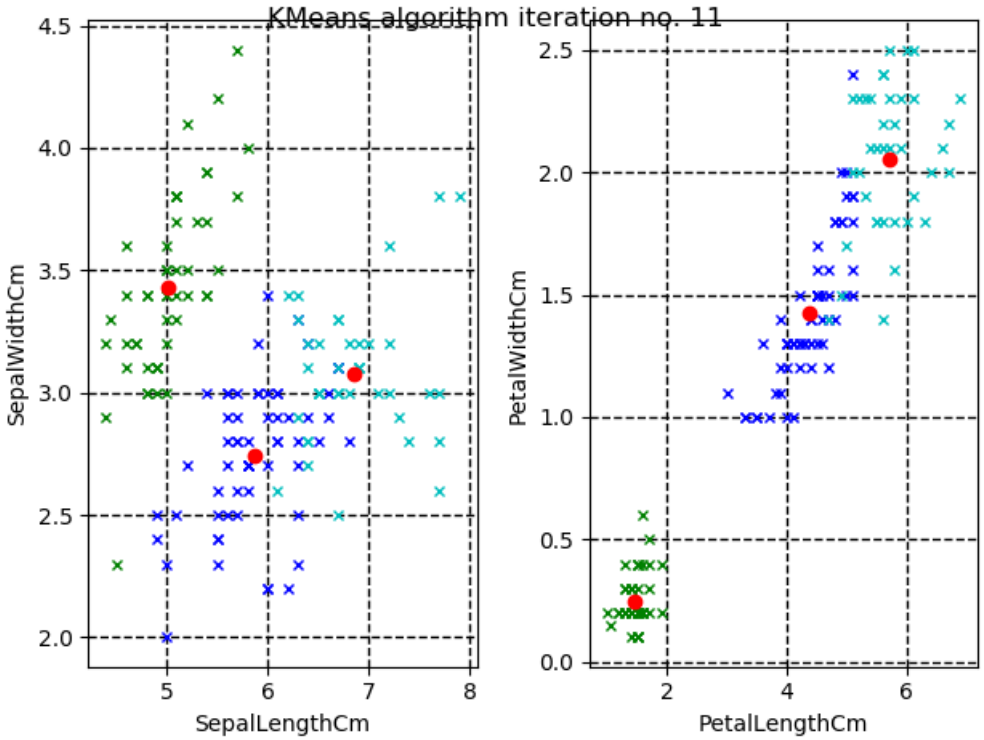
Rysunek 17



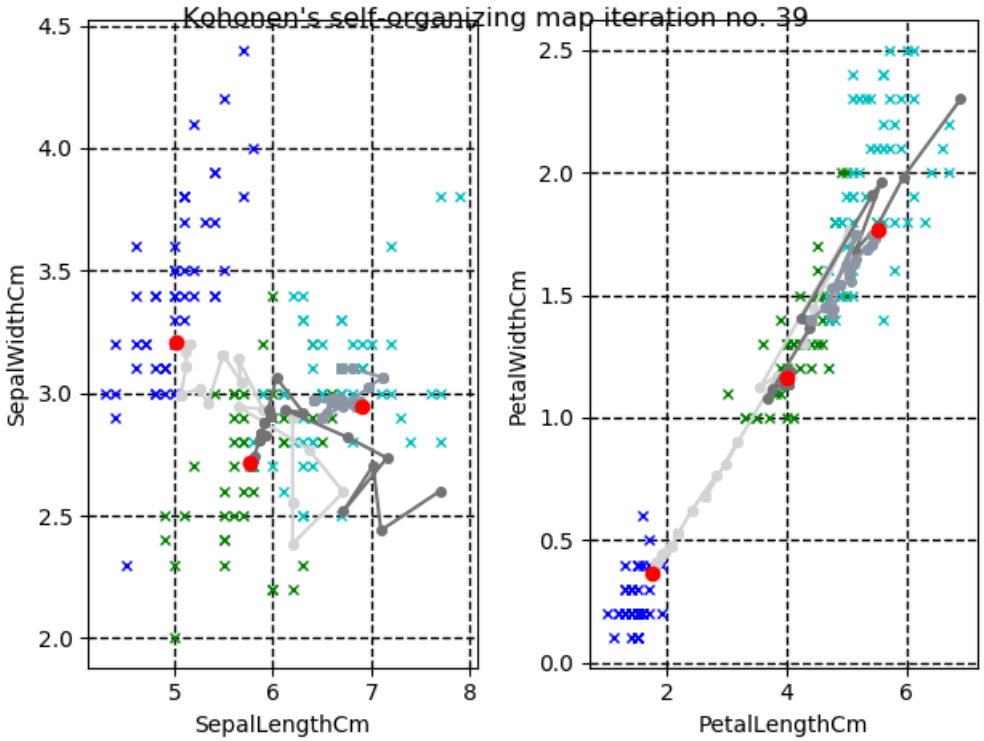
Rysunek 18



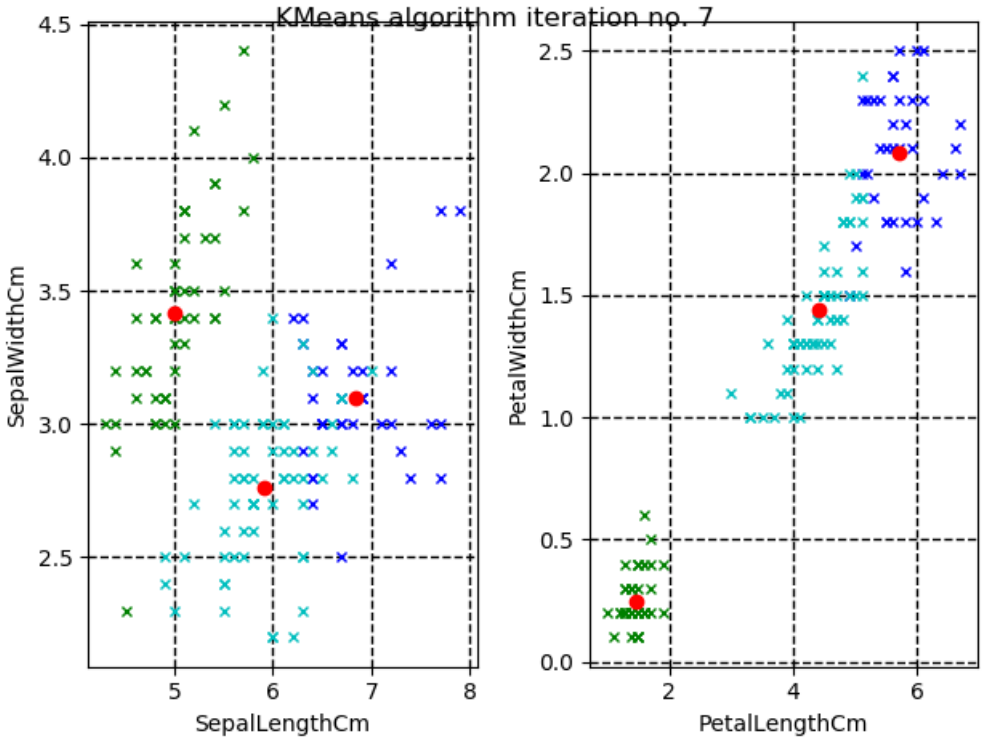
Podpunkt 2.2
zwiększona tolerancja błędu
Rysunek 19



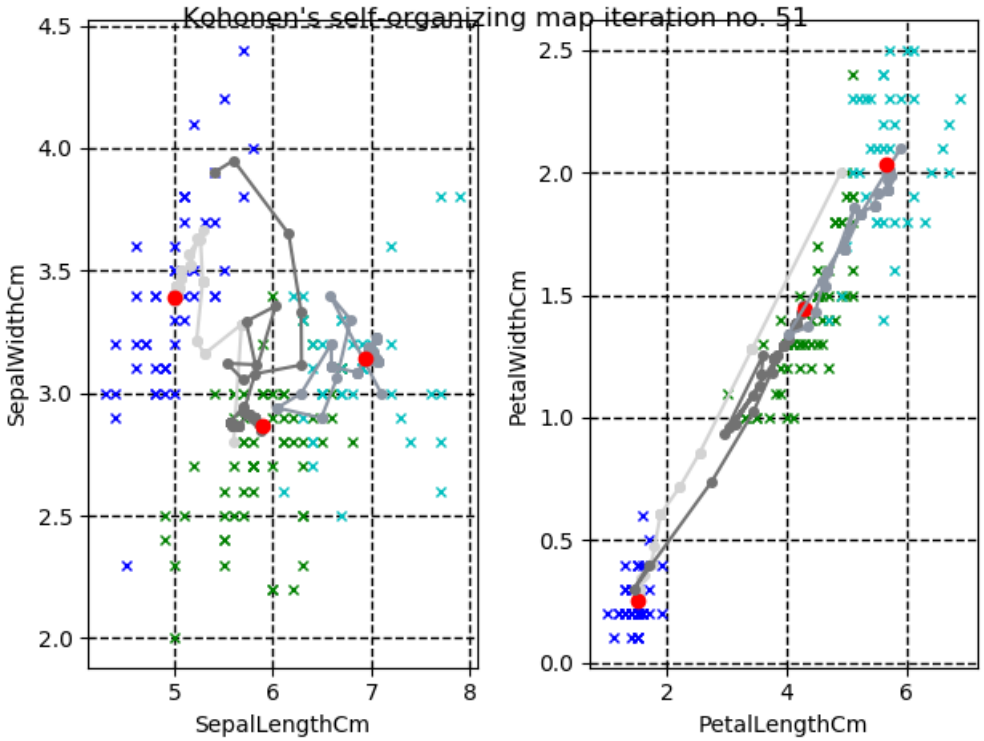
Rysunek 20



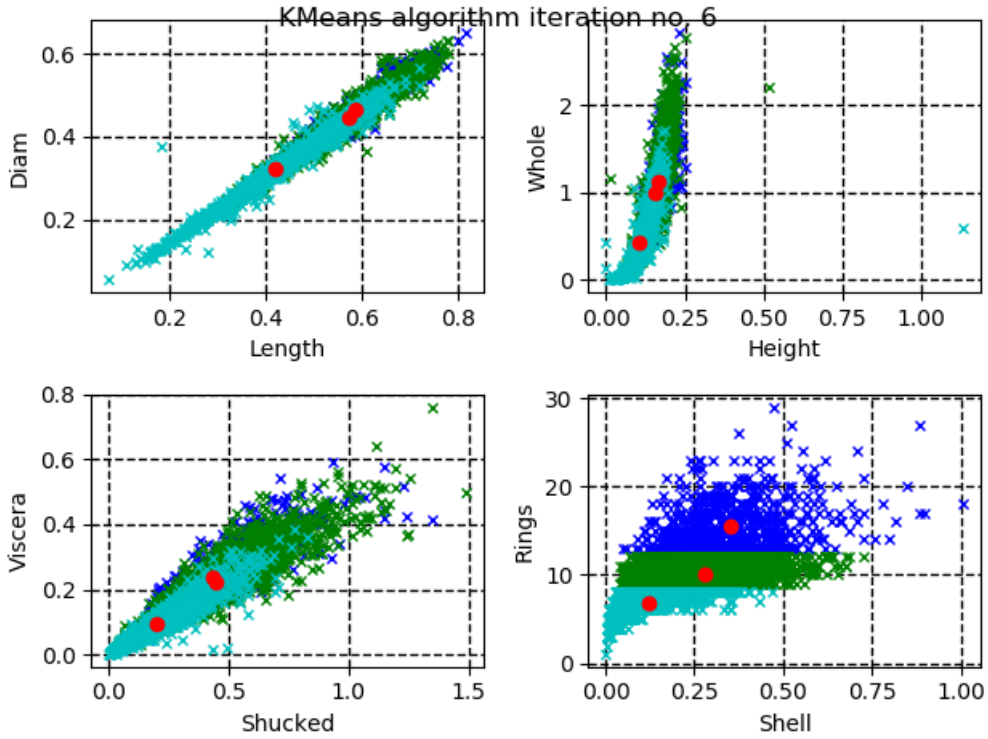
Rysunek 21



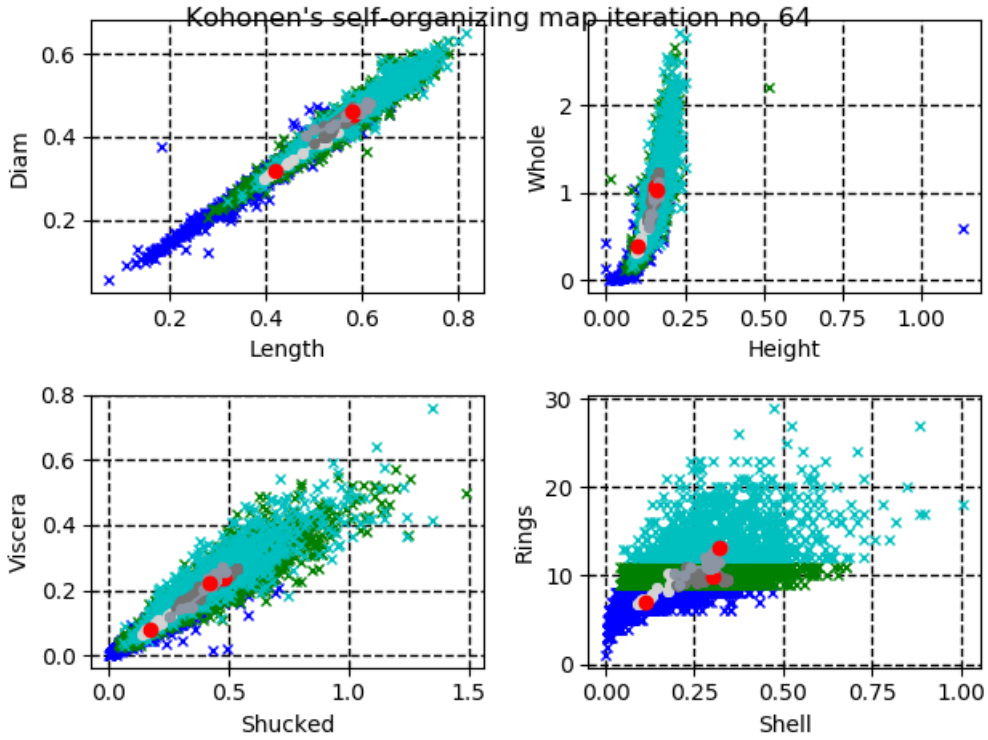
Rysunek 22



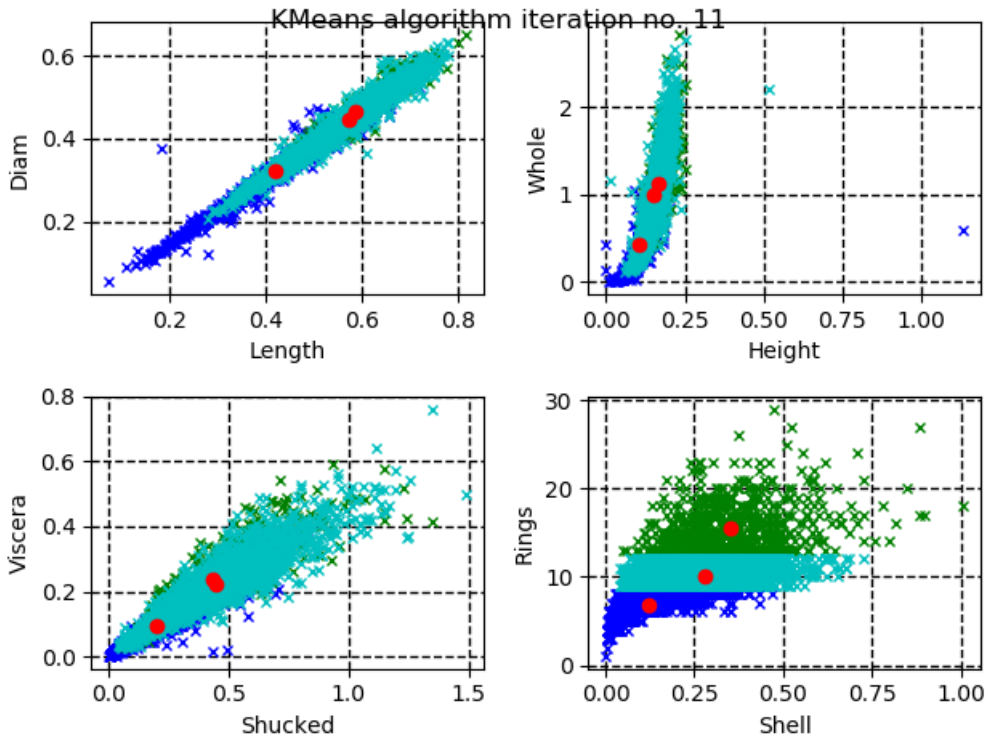
Podpunkt 3.1
Rysunek 23



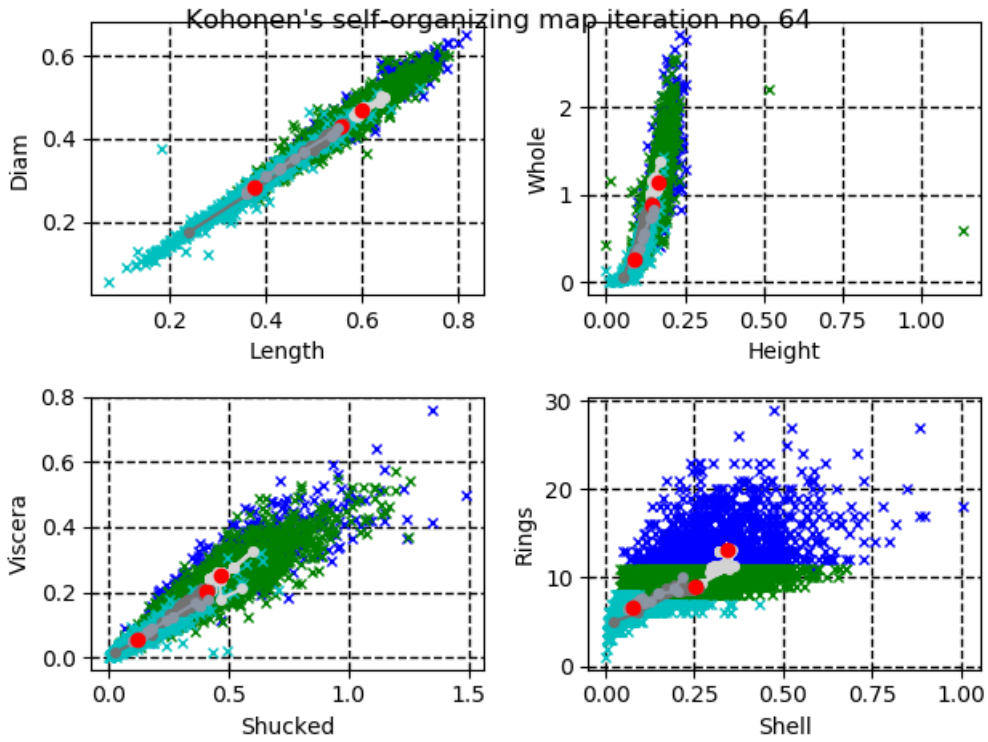
Rysunek 24



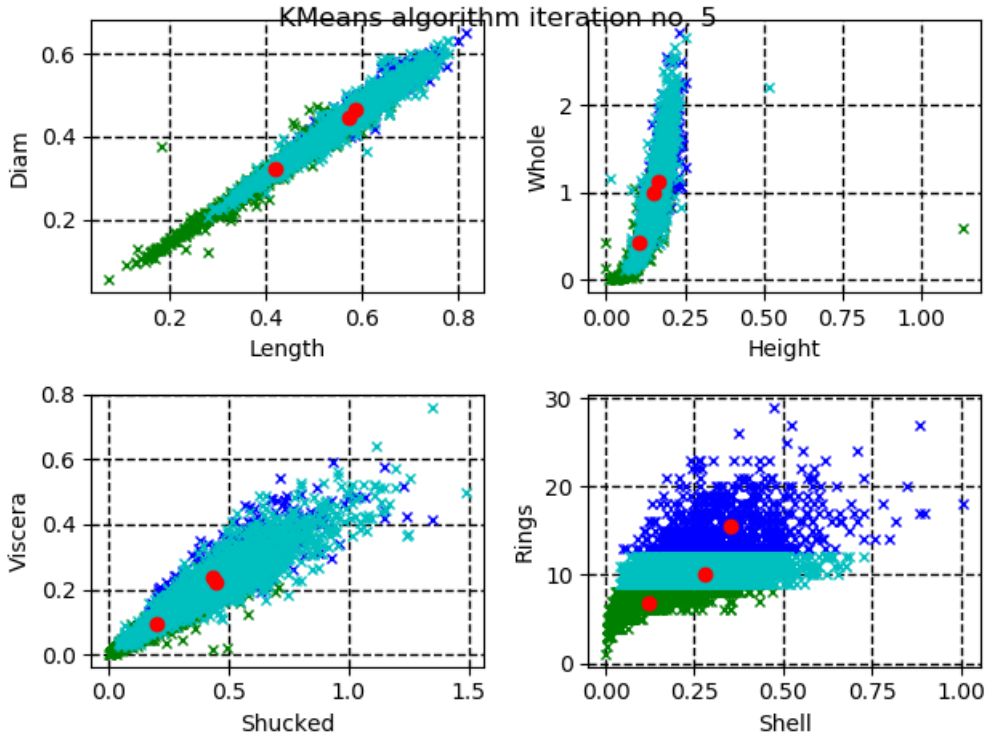
Rysunek 25



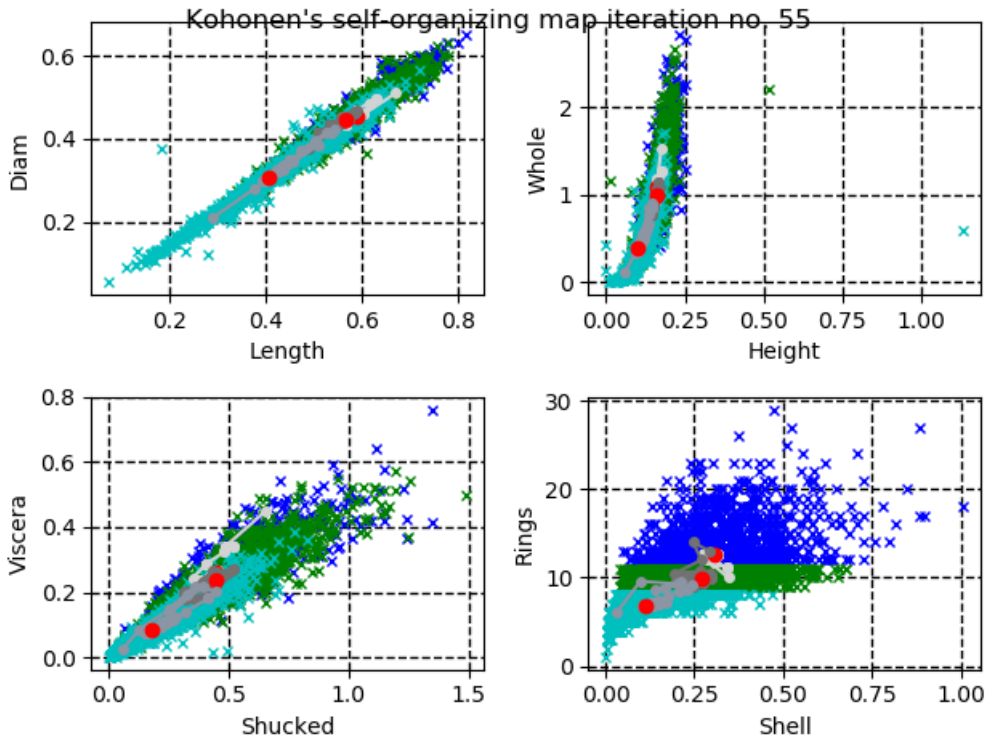
Rysunek 26



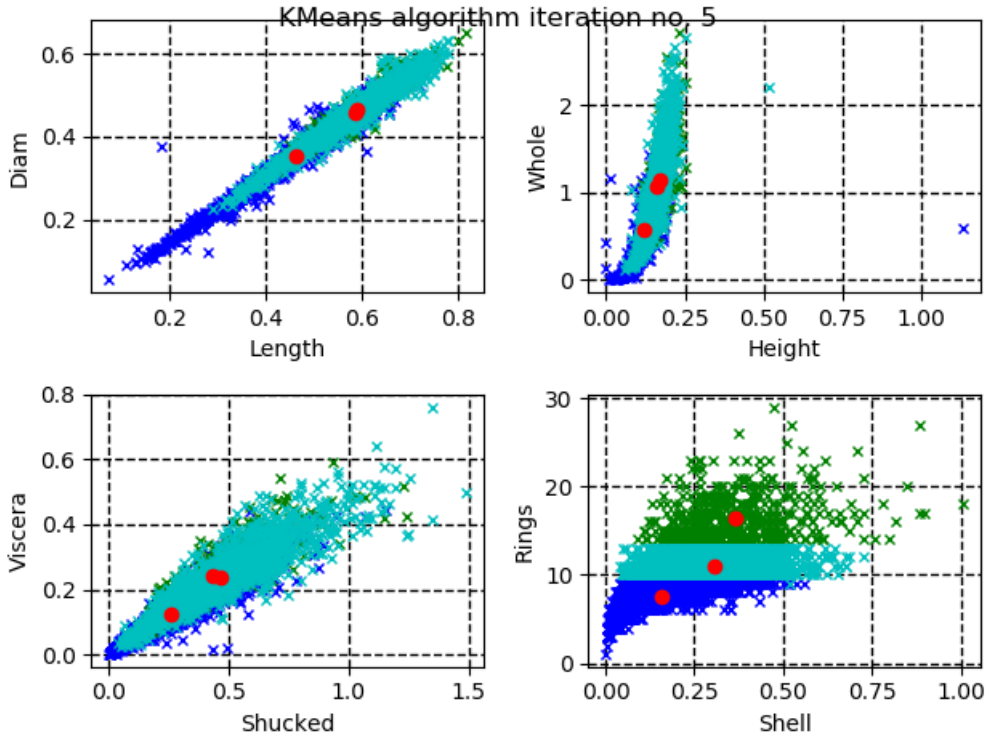
Podpunkt 3.2
zwiększona tolerancja błędu
Rysunek 27



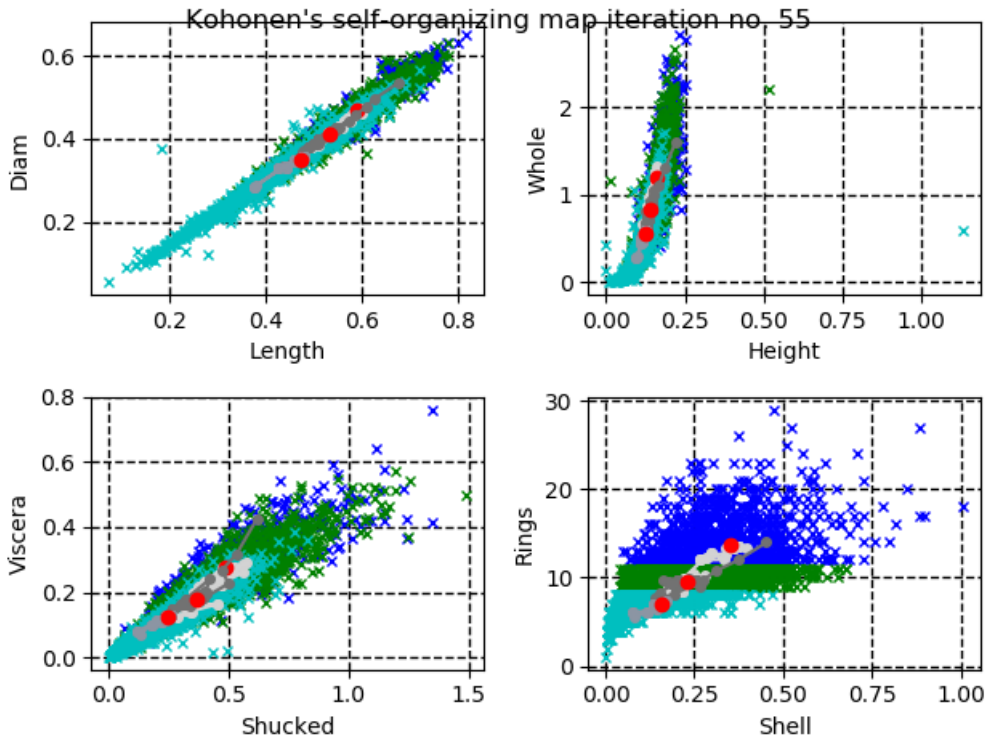
Rysunek 28



Rysunek 29



Rysunek 30



6. Dyskusja

Losowe początkowe rozmieszczenie centroidów/neuronów ma największe

znaczenie dla ilosci potrzebnych iteracji do przetworzenia zestawów danych.

Algorytm K-srednich wymaga o wiele mniej iteracji niż algorytm Kohonena, lecz nie znaczy to że jest szybszy. Czas potrzebny do przetworzenia zestawu o dużej ilosci danych był większy dla k-srednich.

Zwiększenie tolerancji błędu kwantyzacji nie dawało zauważalnych wyników przy algorytmie K-srednich, ale przy algorytmie Kohonena znacznie zmniejszało ilosć potrzebnych iteracji.

Zwiększenie współczynnika lambda lub początkowego promienia sąsiedztwa przy algorytmie Kohonena powodował duże skoki położenia neuronów podczas iteracji, ale klasteryzacja nadal przebiegała poprawnie.

Zmniejszenie początkowego współczynnika nauki dla tego algorytmu drastycznie zmniejszyło długosć skoków neuronów, powodując słabą klasteryzację.

Problem który napotkalismy była szansa, że neuron w sieci kohonena był oddalony od większości punktów danych i jednocześnie był na obrzeżu siatki neuronów, co powodało bardzo powolne przesuwanie (o ile takie istniało).

7. Wnioski

- Ilosć iteracji przy użytych algorytmach jest mocno zależna od początkowych położeń neuronów/centroidów
- Algorytm Kohonena jest szybszy od algorytmu K-srednich dla dużej ilosci danych, chociaż potrzebuje większej ilosci iteracji
- Dobranie zbyt małego początkowego współczynnika nauki dla algorytmu Kohonena może powodować złą klasteryzację
- Duże skoki położenia neuronów w algorytmie Kohonena nie powodują że klasteryzacja będzie źle przebiegać
- Zmiana parametrów początkowych w algorytmie k-srednich nie dawało dużego efektu

Literatura

[1] T. Oetiker, H. Partl, I. Hyna, E. Schlegl. *Nie za krótkie wprowadzenie do systemu LaTeX2e*, 2007, dostępny online.
[2] http://www.ai-junkie.com/ann/som/som1.html.
[3] https://archive.ics.uci.edu/ml/datasets/Iris
[4] https://archive.ics.uci.edu/ml/datasets/seeds
[5] https://archive.ics.uci.edu/ml/datasets/Abalone