Dominik Wróbel

Inżynieria oprogramowania i systemów

Informatyka, II stopień, 2018/19

# Metody eksploracji danych

# Laboratorium 6 – 14.05.2019

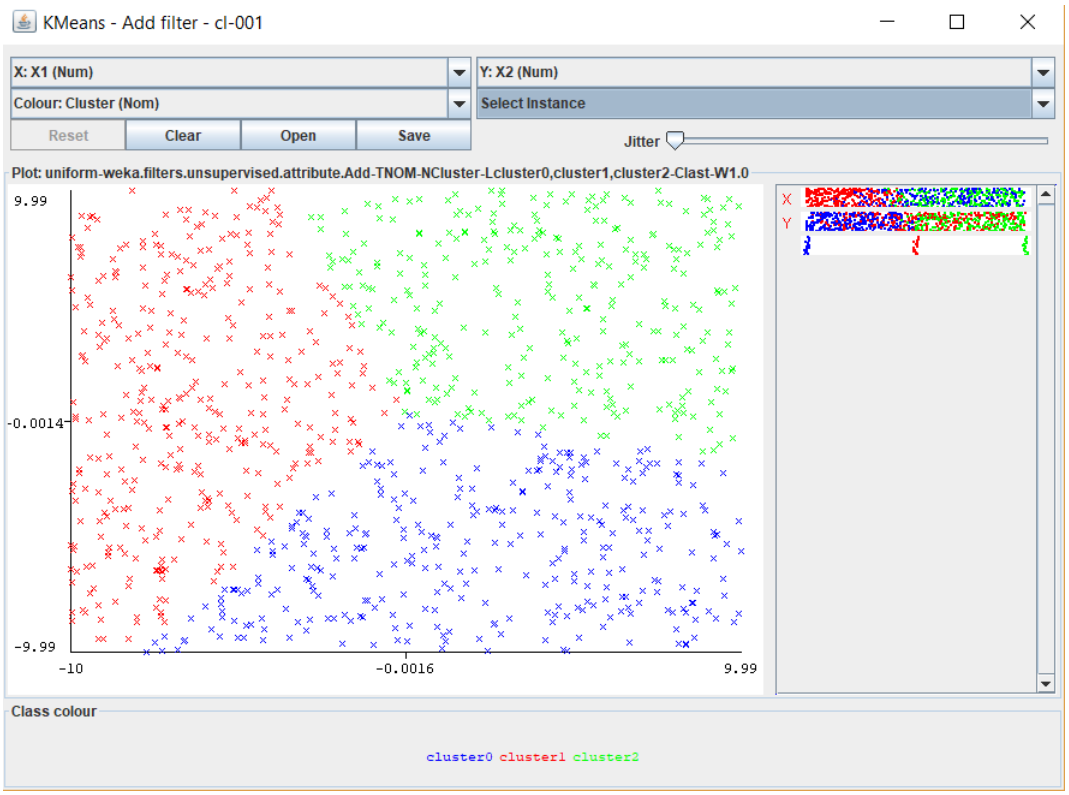
Klasteryzacja

## *6.1 Programowe wywołanie funkcji klasteryzacji Weka*

W tym zadaniu napisano program w języku Java przy użyciu biblioteki weka.jar, który przeprowadza grupowanie na danych z pliku cl-001.arff przy użyciu filtra Add.

```
public class MainWeka6 {

    public static void main(String[] args) {

        System.out.println("This is Java Weka library !");

        try{
            // 6.1.1 - Zaladuj plik cl-001.arff ------------------------------------------------------------------------
            ConverterUtils.DataSource source = new ConverterUtils.DataSource( location: "cl-001.arff");
            Instances data = source.getDataSet();

            // 6.1.2 - Przeprowadz grupowanie za pomoca KMeans ---------------------------------------------------------
            SimpleKMeans cls = new SimpleKMeans();
            cls.setNumClusters(3);
            cls.setPreserveInstancesOrder(true);
            cls.buildClusterer(data);

            // Dodajemy atrybut wyjsciowy, stosujemy filtr Add ---------------------------------------------------------
            Add filter = new Add();
            filter.setAttributeIndex("last");
            int num = cls.numberOfClusters();
            String labels = "cluster0";
            for(int i = 1; i < num; i++){
                labels += ", cluster";
                labels += i;
            }
            filter.setNominalLabels(labels);
            filter.setAttributeName("Cluster");
            filter.setInputFormat(data);
            Instances newData = Filter.useFilter(data, filter);


            // Dodanie etykiet
            int idx = newData.numAttributes()-1;
            for(int i=0;i<newData.numInstances();i++){
                newData.get(i).setValue(idx, cls.clusterInstance(data.get(i)));
            }

            // Wywolanie wizualizacji Weka -----------------------------------------------------------------------------
            MainWeka6.visualize(newData,  title: "KMeans - Add filter - cl-001");
```

W wyniku działania programu otrzymano wizualizację przeprowadzonej klasteryzacji.

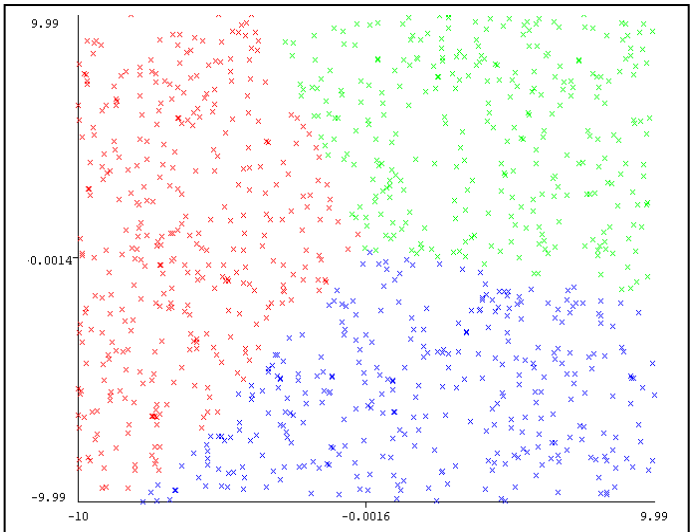

KMeans - Add filter - cl-001
X: X1 (Num)
Y: X2 (Num)
Colour: Cluster (Nom)
Select Instance
Reset
Clear
Open
Save
Jitter
Plot: uniform-weka.filters.unsupervised.attribute.Add-TNOM-NCluster-Lcluster0,cluster1,cluster2-Clast-W1.0
9.99
-0.0014
-9.99
-10
-0.0016
9.99
X
Y
Class colour
cluster0 cluster1 cluster2

## 6.2 Przetestuj wpływ parametrów

W tym zadaniu również przetwarzany jest plik cl-001.arff, ale tym razem testowane są różne wartości parametrów k (liczba grup) oraz różne wartości generatora liczb losowych (seed). Do programu dodano także kod wypisujący informację o środkach klastrów i otrzymanym błędzie.

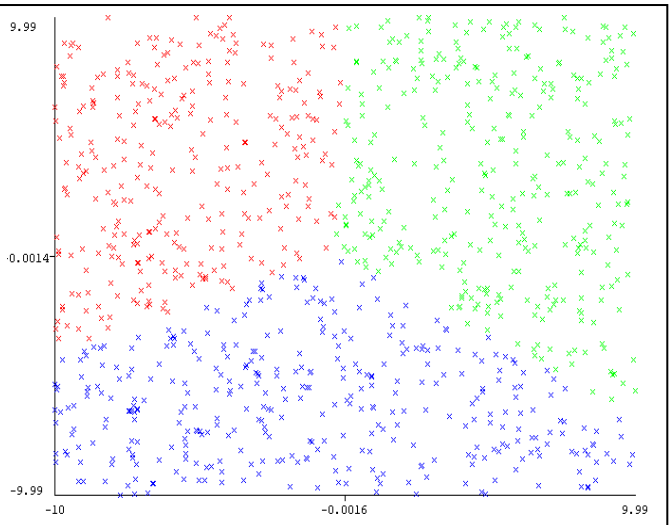
```
try{
    // 6.1.1 - Zaladuj plik cl-001.arff ------------------------------------------------------------------------
    ConverterUtils.DataSource source = new ConverterUtils.DataSource( location: "cl-001.arff");
    Instances data = source.getDataSet();

    // 6.1.2 - Przeprowadz grupowanie za pomoca KMeans ---------------------------------------------------------
    SimpleKMeans cls = new SimpleKMeans();
    cls.setNumClusters(3);
    cls.setSeed(10);
    cls.setPreserveInstancesOrder(true);
    cls.buildClusterer(data);

    // Dodajemy atrybut wyjsciowy, stosujemy filtr Add ---------------------------------------------------------
    Add filter = new Add();
    filter.setAttributeIndex("last");
    int num = cls.numberOfClusters();
    String labels = "cluster0";
    for(int i = 1; i < num; i++){
        labels += ", cluster";
        labels += i;
    }
    filter.setNominalLabels(labels);
    filter.setAttributeName("Cluster");
    filter.setInputFormat(data);
    Instances newData = Filter.useFilter(data, filter);


    // Dodanie etykiet
    int idx = newData.numAttributes()-1;
    for(int i=0;i<newData.numInstances();i++){
        newData.get(i).setValue(idx, cls.clusterInstance(data.get(i)));
    }

    // Wyniki dla k-means
    Instances centroids = cls.getClusterCentroids();
    for(int i=0;i<centroids.numAttributes();i++){
        System.out.print(centroids.attribute(i));
        System.out.print(",");
    }
    System.out.print("\n");
    for(int i=0;i<centroids.numInstances();i++){
        System.out.println(centroids.get(i));
    }
    System.out.printf(Locale.US, format: "Error:%f",cls.getSquaredError());

    // Wywolanie wizualizacji Weka -----------------------------------------------------------------------------
    MainWeka6.visualize(newData, title: "KMeans - Add filter - cl-001");
```

- Dla k równego 3, przetestowano różne wartości parametru seed generatora liczb losowych



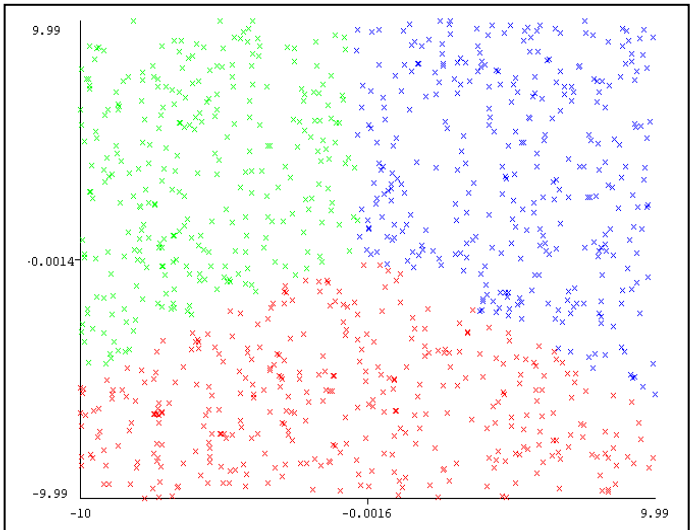

k=3, seed=10, Error = 67.271506

| X | Y |
|---|---|
| 2.157022 | -5.633541 |
| -6.377598 | 0.868186 |
| 4.233062 | 5.053473 |



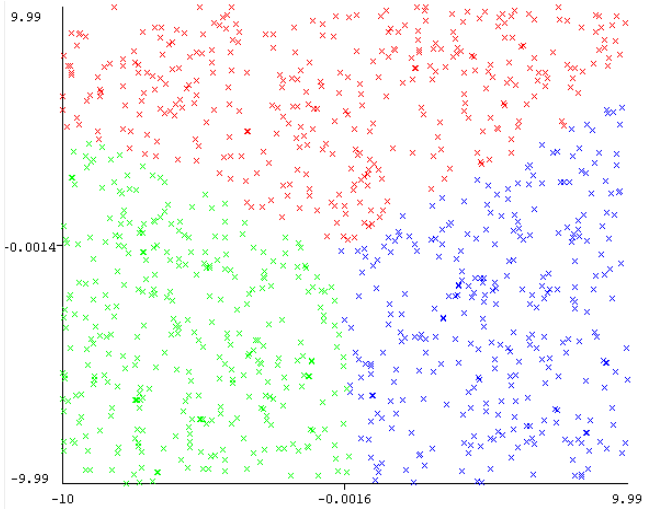

k=3, seed=50, Error = 67.258977

| X | Y |
|---|---|
| -1.14128 | -6.137768 |
| -5.619137 | 3.691901 |
| 5.176742 | 3.305419 |




k=3, seed=100, Error = 67.286249

| X | Y |
|---|---|
| 5.099409 | 3.620232 |
| -0.77498 | -6.074754 |
| -5.785441 | 3.446899 |

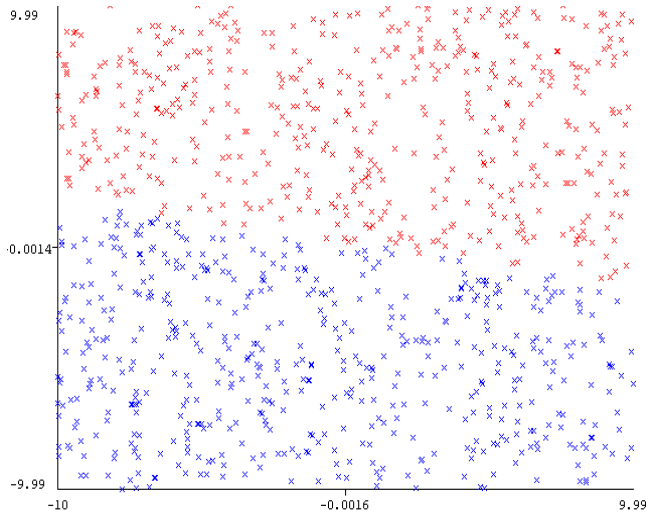



k=3, seed=2, Error = 65.693046

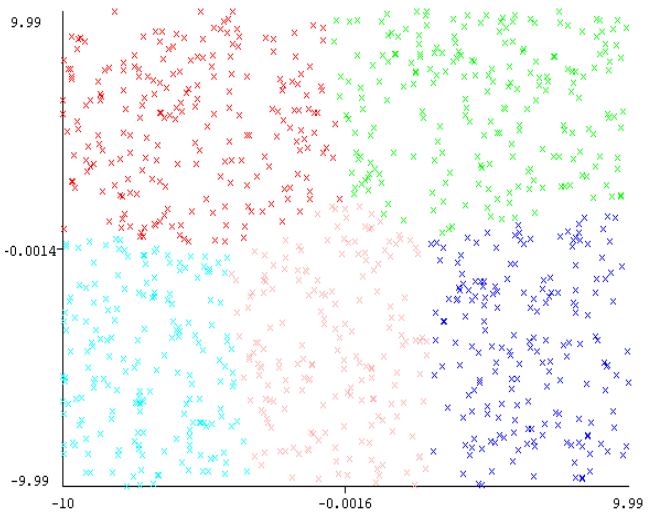
| X | Y |
|---|---|
| 5.42712 | -3.087031 |
| -0.271195 | 6.279937 |
| -5.502327 | -3.759238 |

Eksperyment potwierdza, że algorytm jest bardzo wrażliwy na parametr seed, zmiana tego parametru spowodowała wygenerowanie środków o znacznie różnych wartościach.

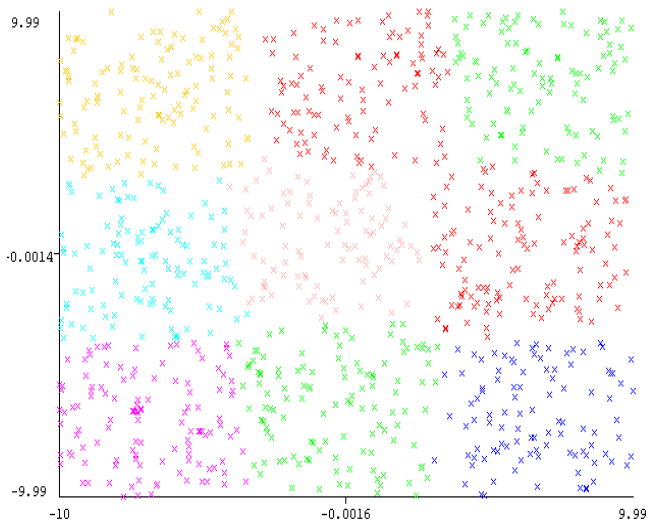
- Dla seed=10 przetestowano różne wartości parametru k



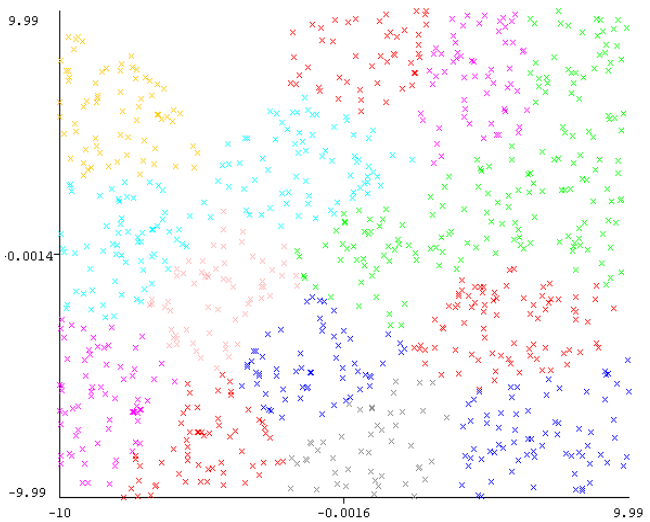

k=2, seed=10, Error = 105.429206




k=5, seed=10, Error = 35.345713




k=9, seed=10, Error = 17.300230



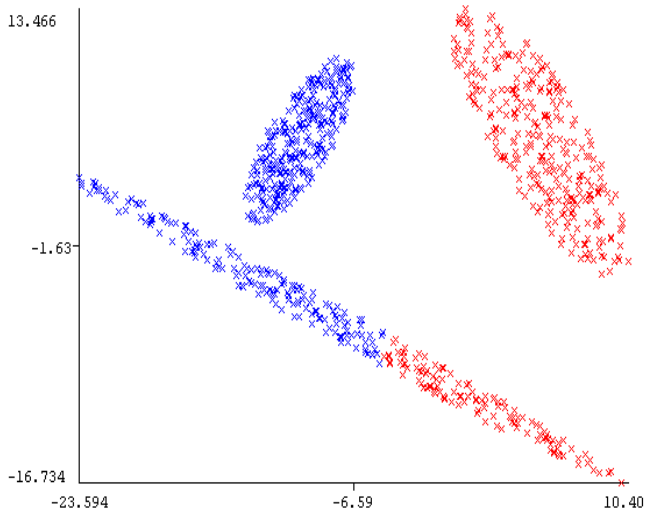

k=16, seed=10, Error = 10.389671

Eksperyment pokazuje duży wpływ liczby grup na wartość błędu.

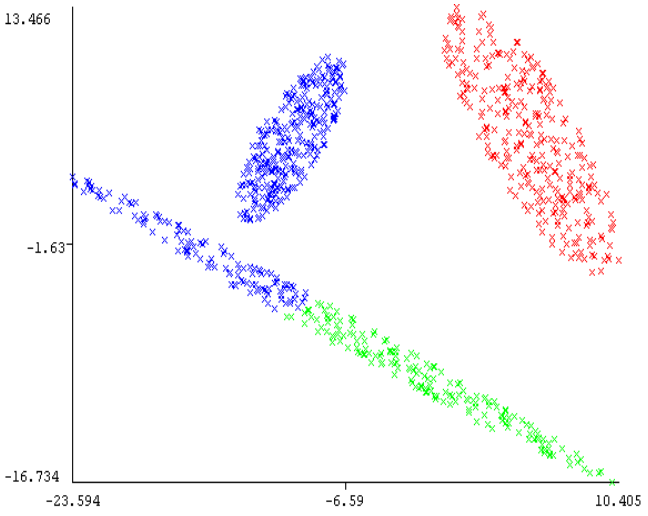
## *6.3 Przetwarzamy pliki za pomocą k-means*

W tym zadaniu uruchamiano program dla różnych plików z danymi i dla różnej liczby grup (zależnie od danych). Wyniki wizualizacji przedstawiono poniżej.

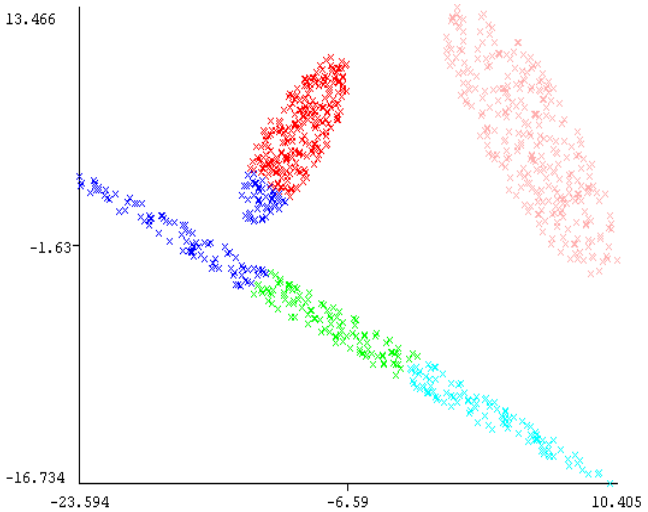
- Plik cl-002.arff



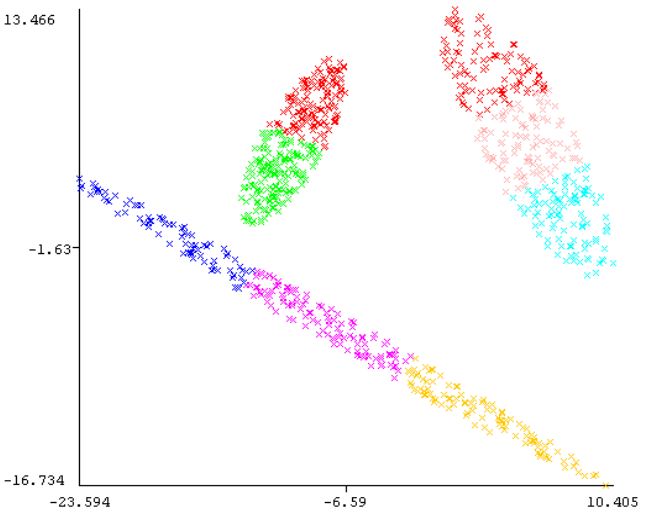

k=2, seed=10, Error = 56.508932



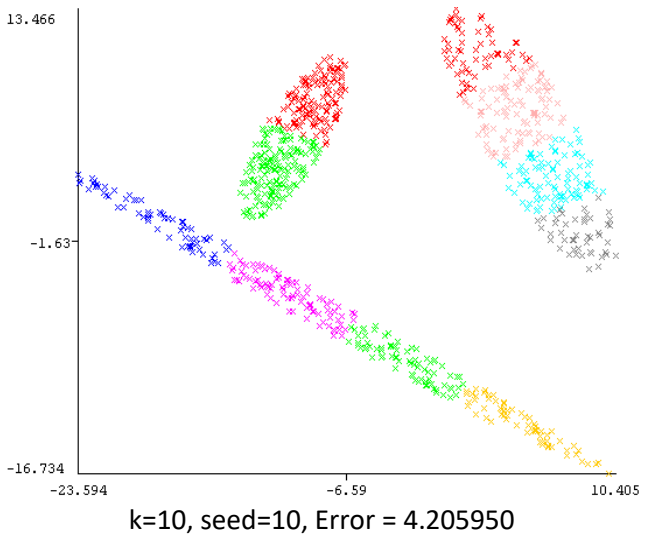

k=3, seed=10, Error = 25.897798




k=5, seed=10, Error = 14.194627



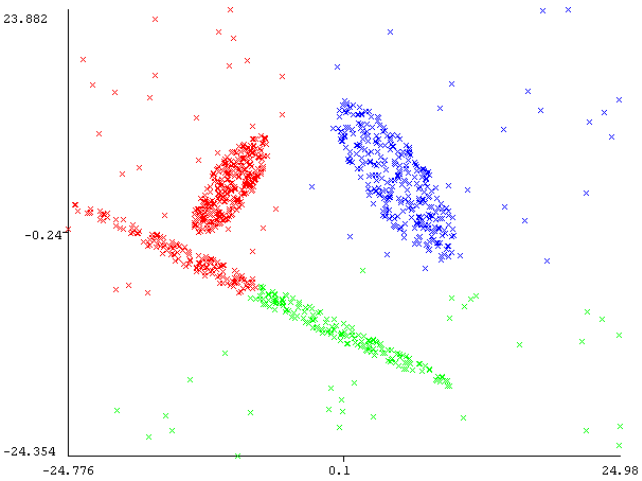

k=8, seed=10, Error = 6.245052




k=10, seed=10, Error = 4.205950

Wraz ze wzrostem parametru k błąd maleje.

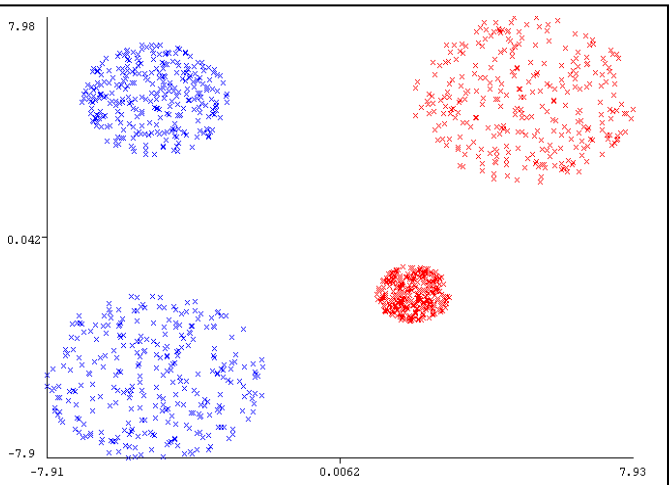
- Plik cl-003.arff



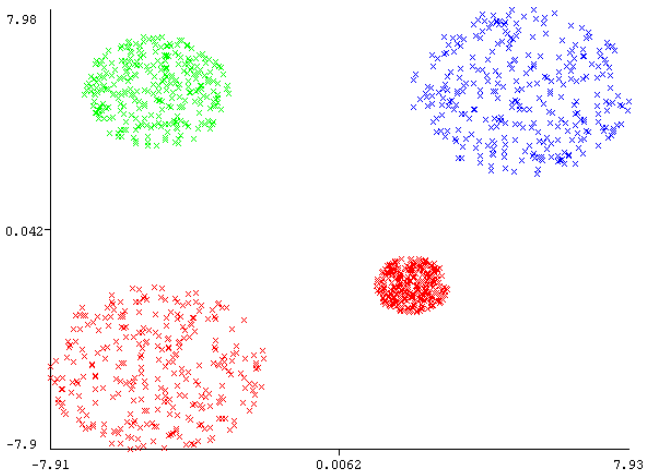

k=3, seed=10, Error = 18.767046

| X | Y |
|---|---|
| 5.516921 | 5.504307 |
| -11.643133 | 2.821538 |
| 0.730146 | -11.83098 |

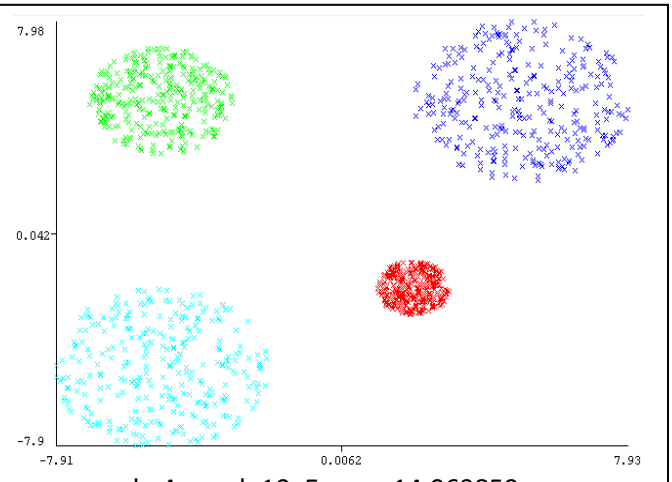
- Plik cl-004.arff




k=2, seed=10, Error = 109.647491




k=3, seed=10, Error = 48.670713

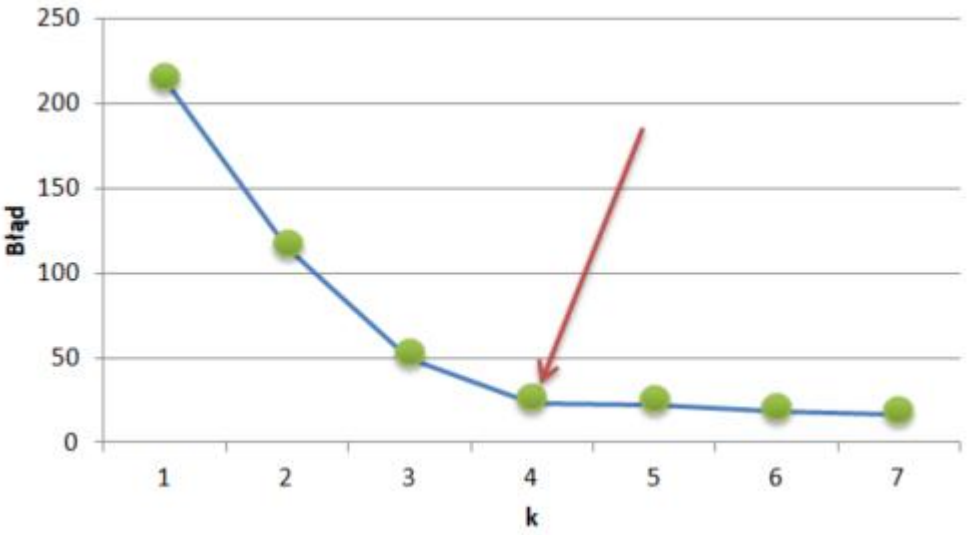


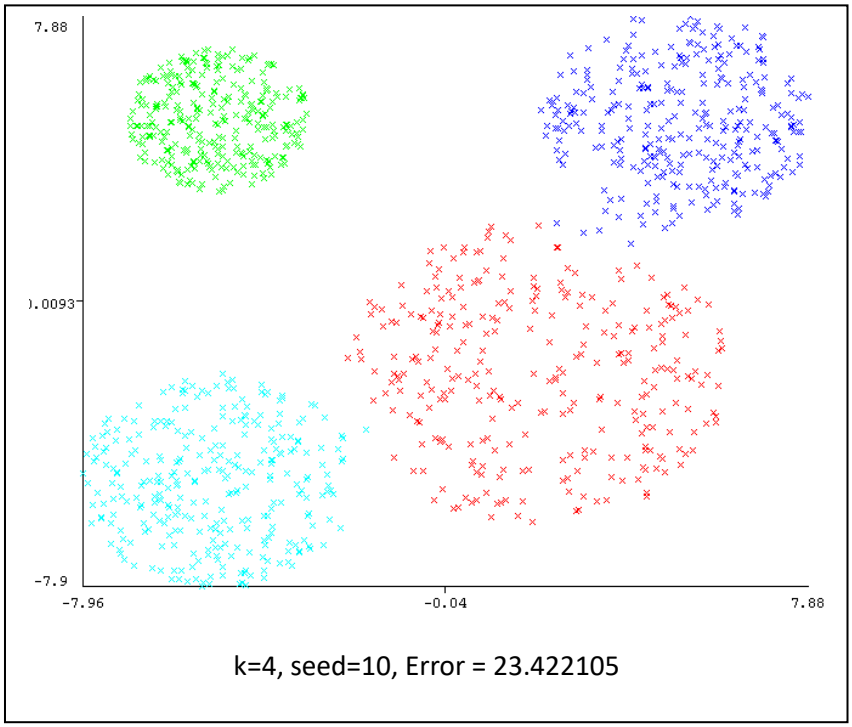

k=4, seed=10, Error = 14.069859

- Plik cl-005.arff

W tym punkcie dobierana jest najlepsza wartość k. Istnieje wiele metod pozwalających na wyznaczenie optymalnej wartości k. Jedną z nich jest tzw. *elbow metod*, która pozwala na wyznaczenie odpowiedniej liczby klastrów na podstawie wykresu zależności błędu od k. Za optymalną liczbę klastrów przyjmuje się taką, która na wykresie odpowiada punktowi po którym dla większej liczby klastrów klasteryzacja nie przynosi już znaczącej poprawy wartości błędu. Przy pomocy tej metody wyznaczono najlepsze wartości parametru k dla kolejnych plików z danymi.

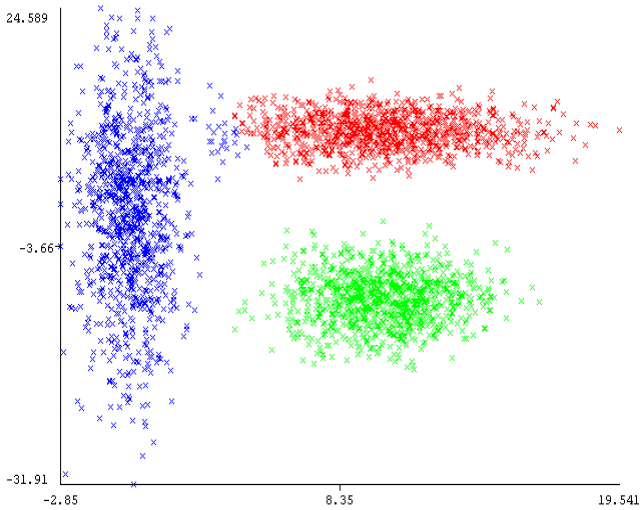
Przykładowo dla pliku cl-005.arff najlepsze k wyznaczone na podstawie wykresu wynosi 4.



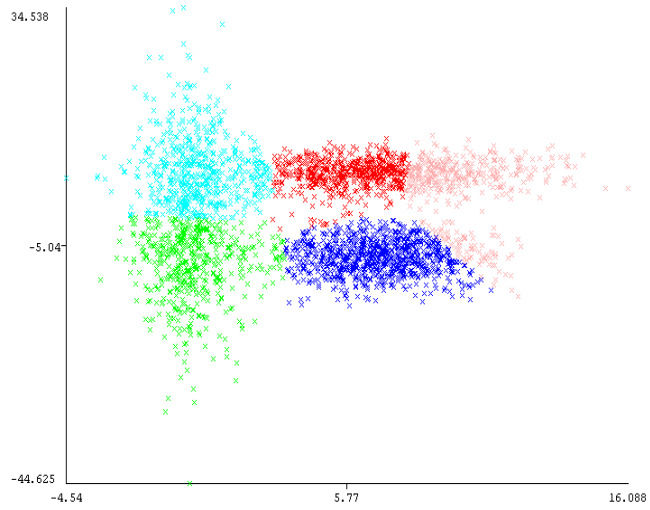




k=4, seed=10, Error = 23.422105

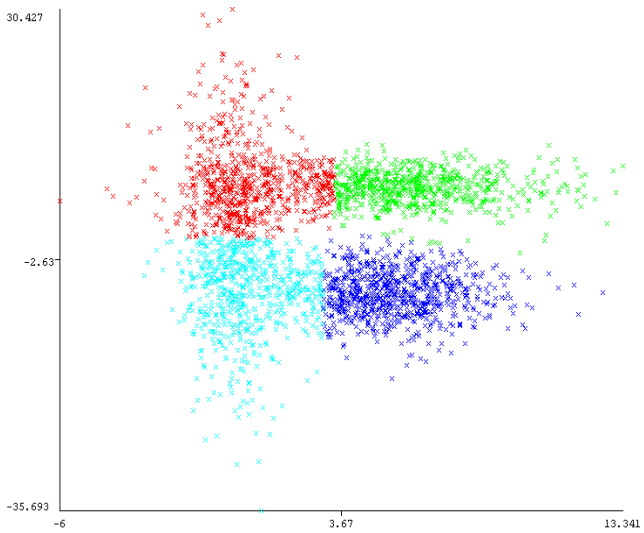
- Pliki cl-006.arff – cl-010.arff



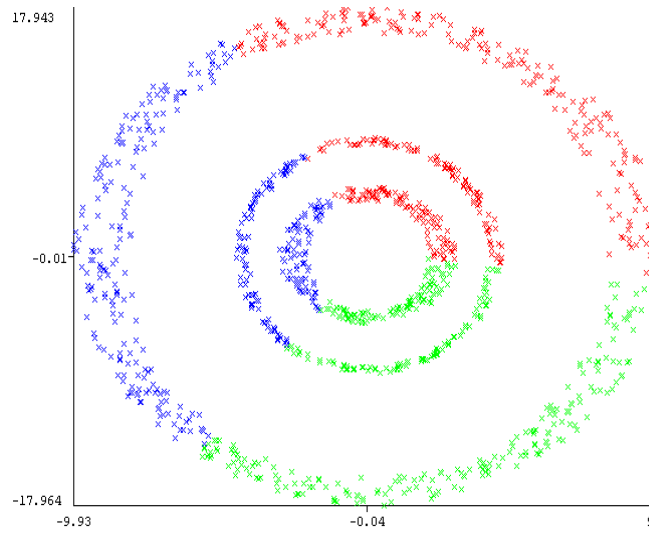
cl-006.arff: k=3, seed=10, Error = 59.439722



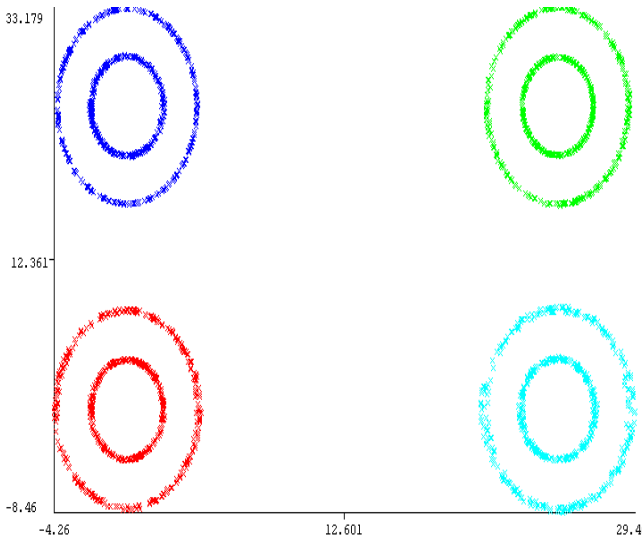
cl-007.arff: k=5, seed=10, Error = 23.356125



cl-008.arff: k=4, seed=10, Error = 31.926304



cl-009.arff: k=3, seed=10, Error = 65.289139

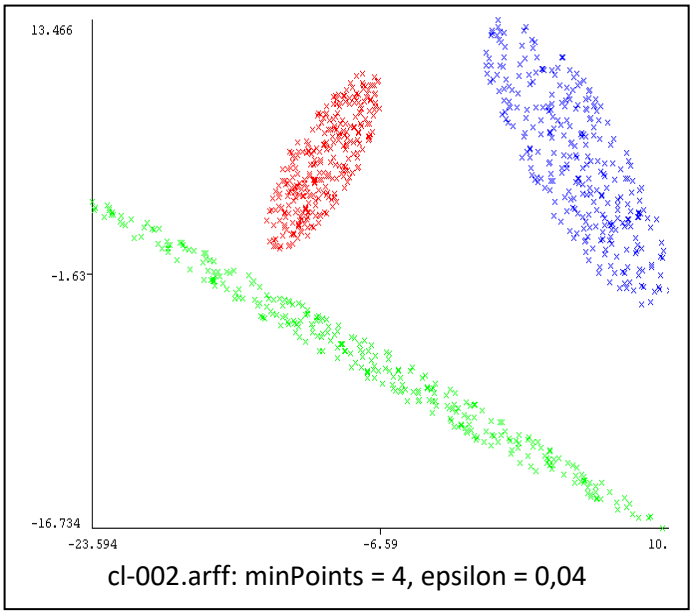


cl-010.arff: k=4, seed=10, Error = 40.452607

## *6.4 Algorytm DBSCAN*

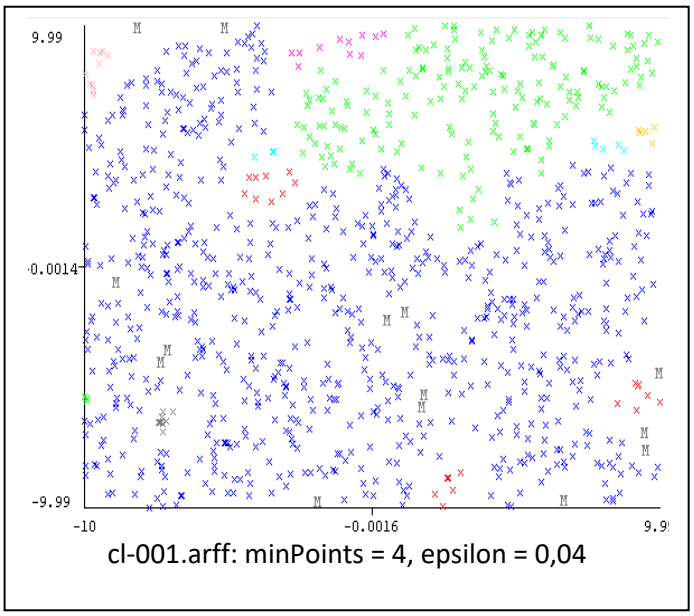
W tym zadaniu do klasteryzacji użyto algorytm DBSCAN, działanie tego algorytmu opiera się na obliczaniu odległości pomiędzy sąsiednimi punktami oraz ignorowaniu niewielkich pojedynczych grup punktów.

- Plik cl-002.arff




cl-002.arff: minPoints = 4, epsilon = 0,04

- Plik cl-001.arff

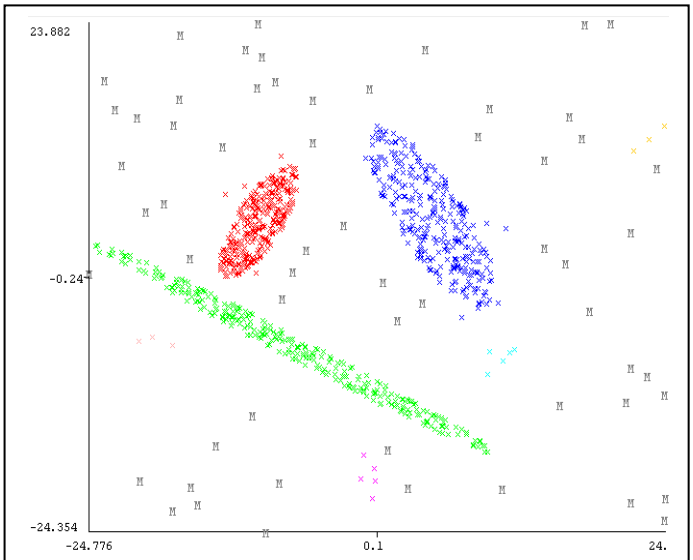



cl-001.arff: minPoints = 4, epsilon = 0,04

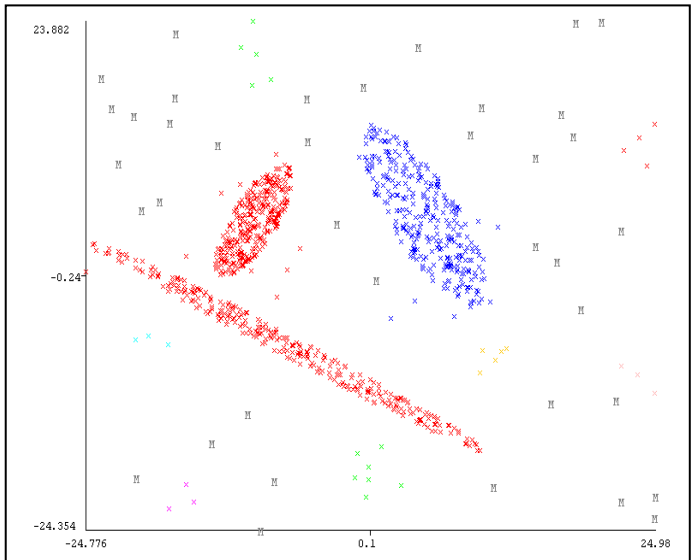
Wynik uzyskany dla tych danych wynika ze sposobu działania algorytmu. Algorytm przypisał do osobnych klastrów małe grupki punktów (np. kolor czerwony, pomarańczowy), które oddalone są od innych punktów o więcej niż określoną w algorytmie odległość. Ponadto niektórym punktom nie został przypisany żaden klaster (szara litera M na wizualizacji) ze względu na parametr minPoints, który określa minimalną potrzebą liczbę punktów do utworzenia klastra.

- Plik cl-003.arff

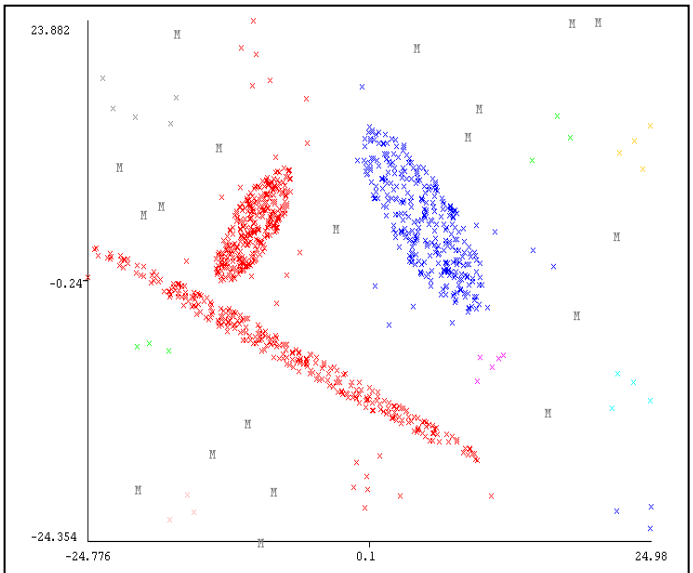
Wyjątki pojawiają się ponieważ nie każdemu z punktów został przypisany klaster. Dla tych danych zbadano wpływ parametru epsilon na wynik działania klasteryzacji.



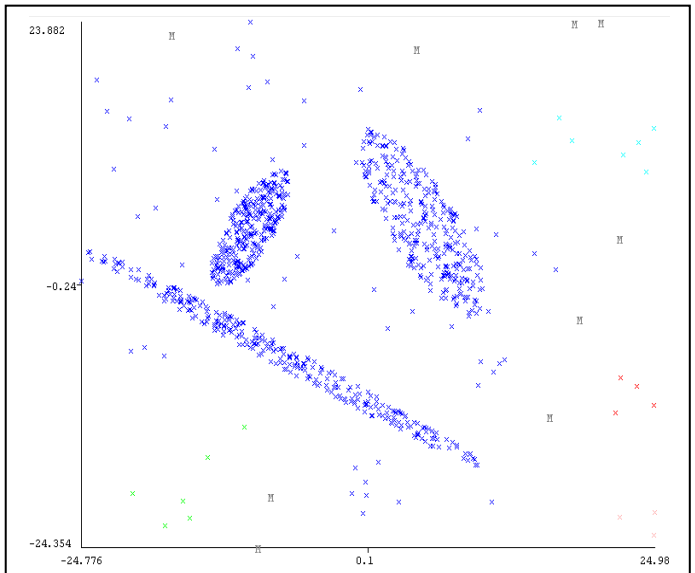
cl-003.arff: minPoints = 4, epsilon = 0,04



cl-003.arff: minPoints = 4, epsilon = 0,06



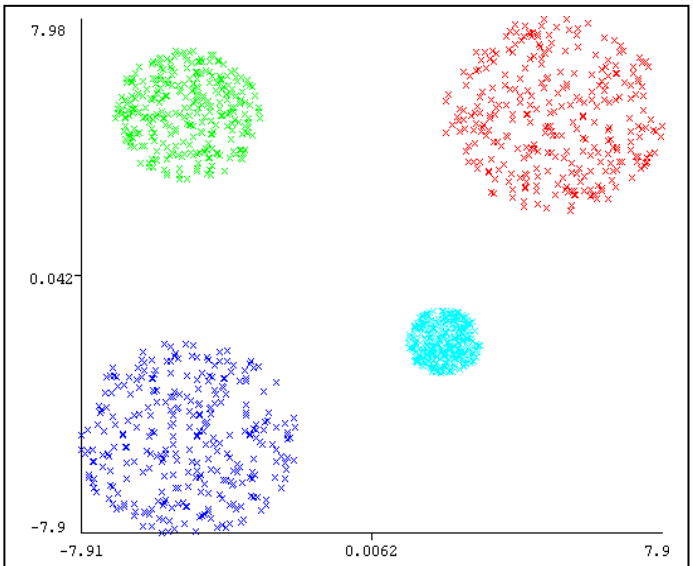
cl-003.arff: minPoints = 4, epsilon = 0,08



cl-003.arff: minPoints = 4, epsilon = 0,1

Zgodnie z oczekiwaniami wzrost parametru epsilon powoduje przypisywanie większej ilości punktów do danego klastra, a także tworzenia się większej ilości klastrów.

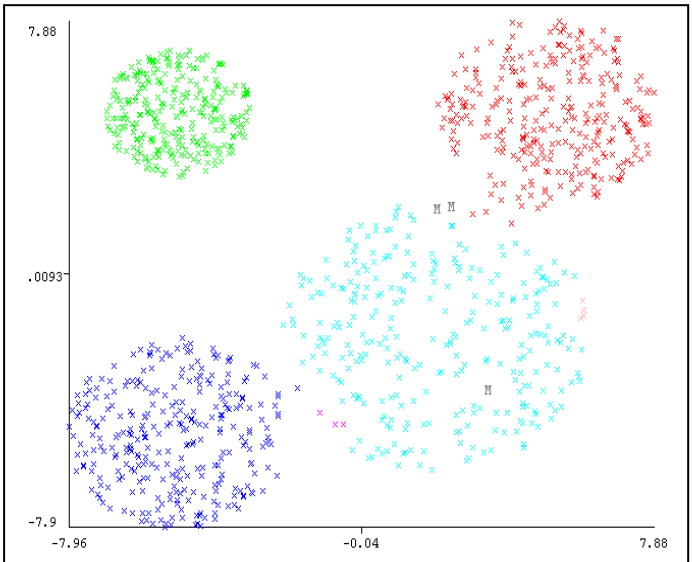
- Plik cl-004.arff



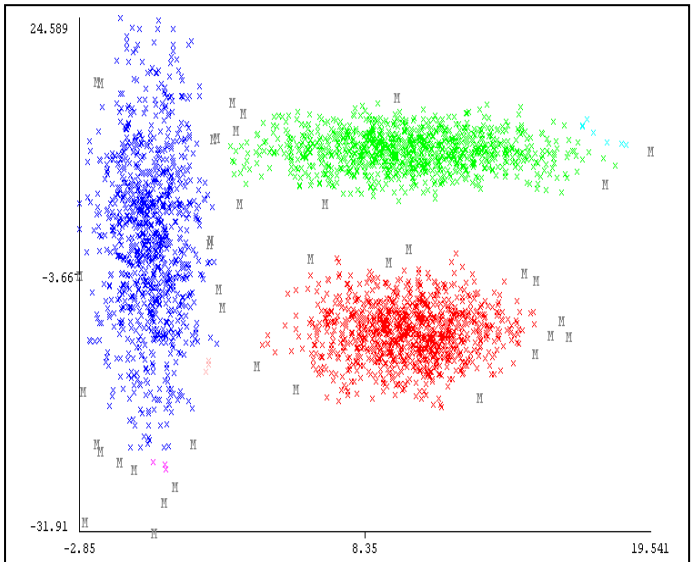
cl-004.arff: minPoints = 4, epsilon = 0,04

- Plik cl-005.arff, cl-006.arff

Dla tego i kolejnych plików podjęto próbę dobrania jak najlepszej wartości parametru epsilon. Istnieje algorytm, który pozwala wyznaczyć wartość epsilon na podstawie tzw. wykresu k-dist. Ze względu na złożoność tworzenia takiego wykresu parametr epsilon dobrano doświadczalnie obserwując wyniki na wykresach.

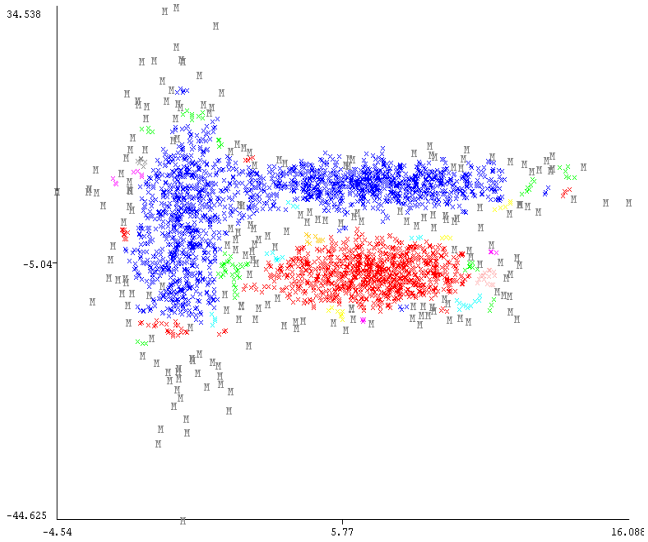


cl-005.arff: minPoints = 3, epsilon = 0,039

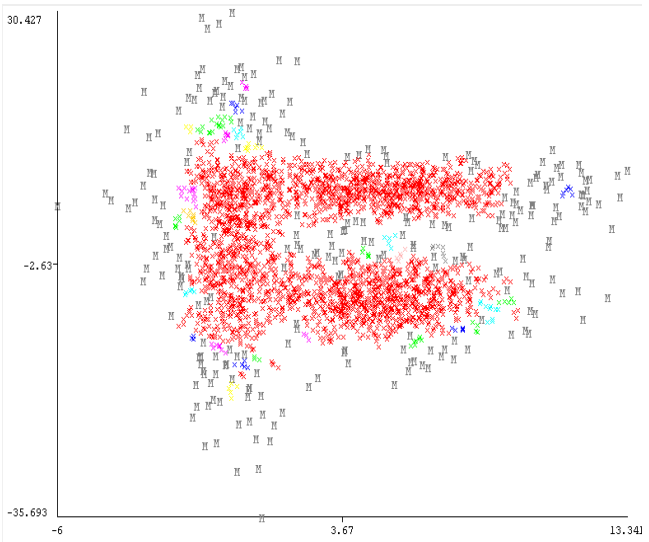


cl-006.arff: minPoints = 3, epsilon = 0,03

- Pliki cl-007.arff, cl-008.arff



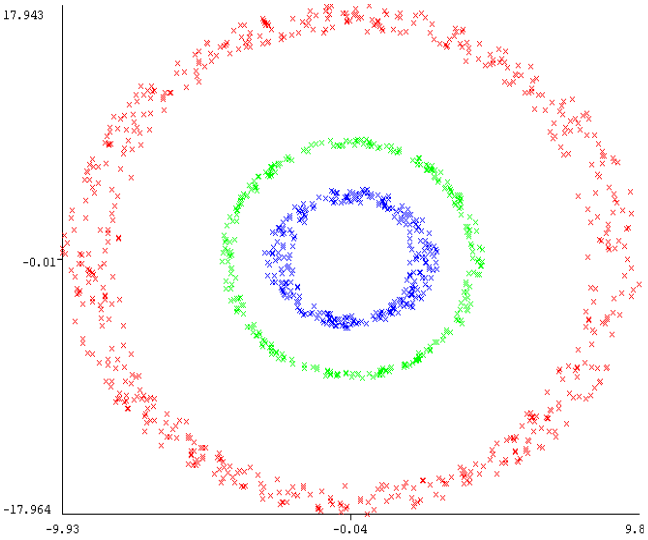

cl-007.arff: minPoints = 3, epsilon = 0,0124



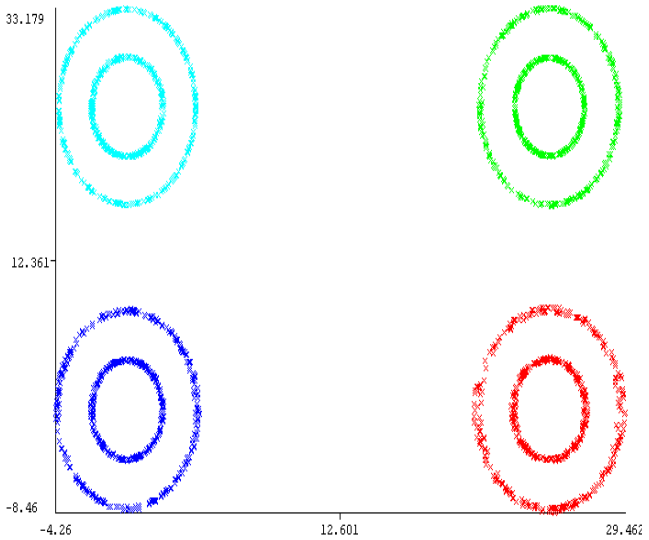

cl-008.arff: minPoints = 3, epsilon = 0,0124

Wyniki dla tych dwóch plików są zbliżone, algorytm DBSCAN nie radzi sobie zbyt dobrze dla tego rodzaju danych, punkty znajdują się bardzo blisko siebie ‘w środku’ zbioru, a dalej od siebie na jego obrzeżach.

- Pliki cl-009.arff, cl-010.arff




cl-009.arff: minPoints = 3, epsilon = 0,04

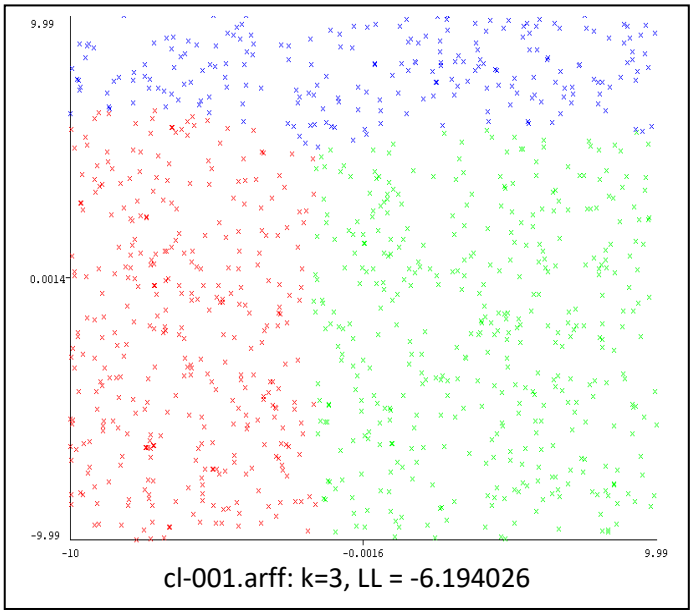



cl-010.arff: minPoints = 3, epsilon = 0,06

## *6.5 Przetwarzamy pliki*

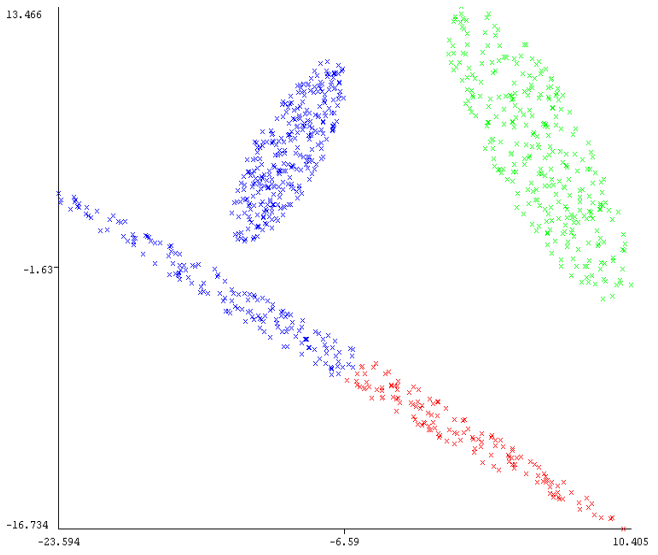
W tym zadaniu do klasteryzacji użyto algorytmu EM bazującego na rozkładzie Gaussa.

- Plik cl-001.arff

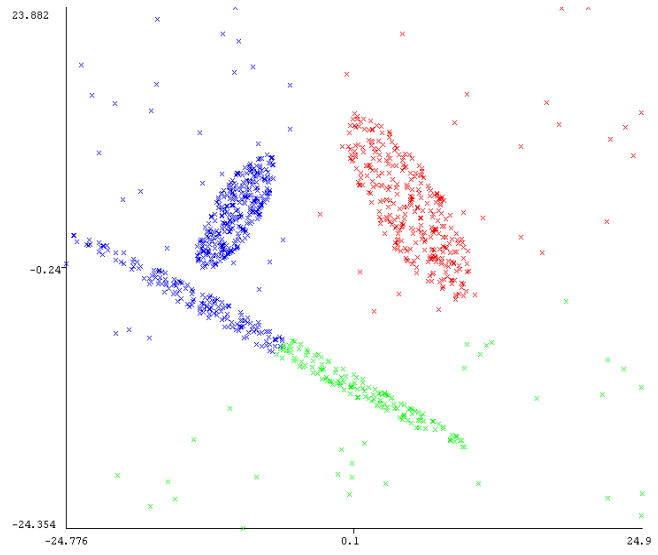



cl-001.arff: k=3, LL = -6.194026

- Pliki cl-002.arff i cl-003.arff

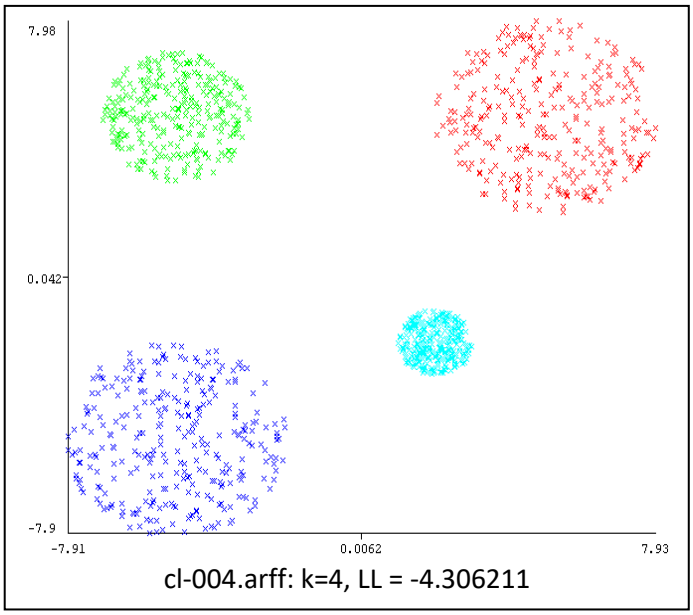



cl-002.arff: k=3, LL = -6.435150

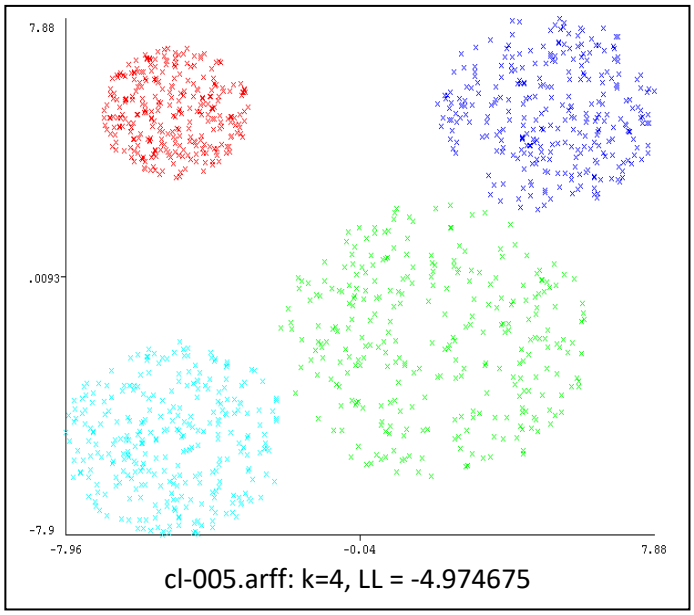



cl-003.arff: k=3, LL = -6.194026

- Pliki cl-004.arff i cl-005.arff

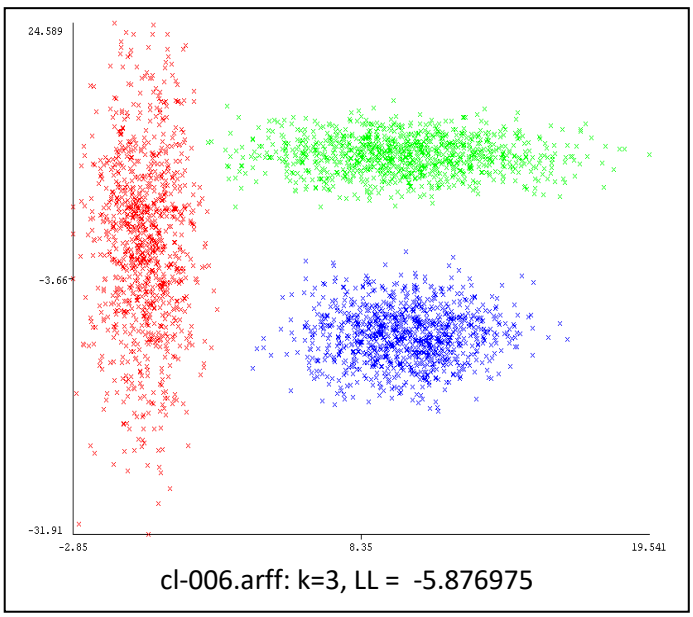



cl-004.arff: k=4, LL = -4.306211

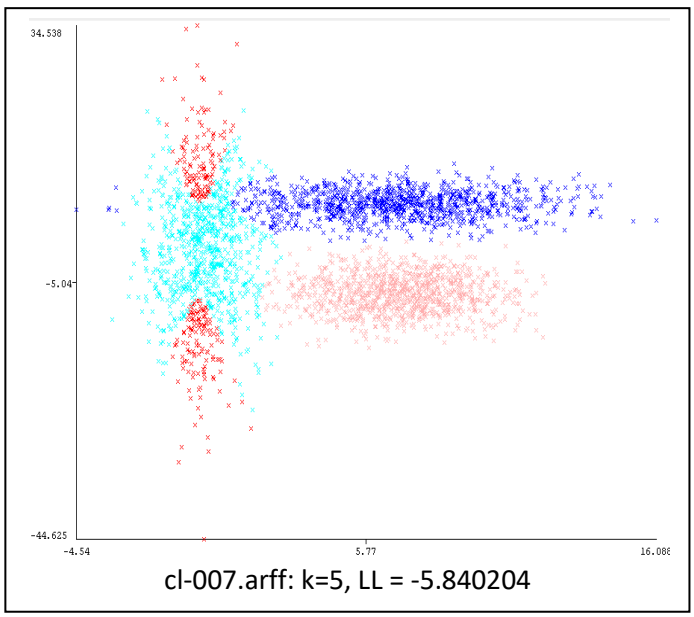



cl-005.arff: k=4, LL = -4.974675

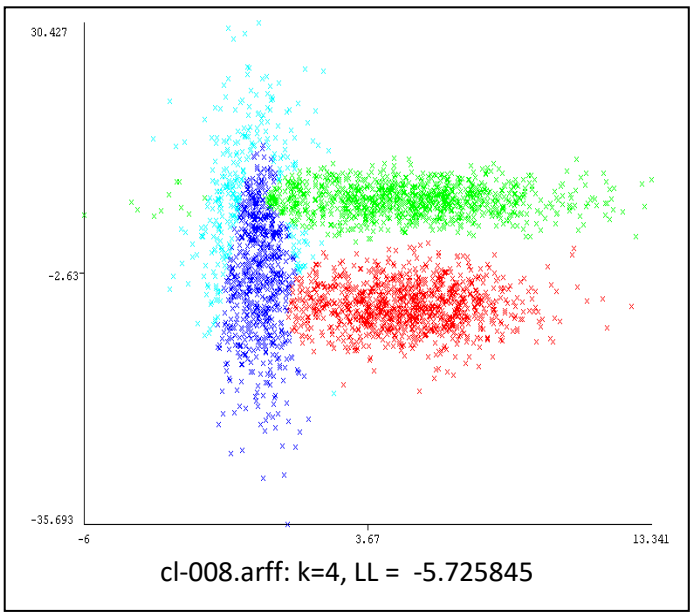
- Pliki cl-006.arff, cl-007.arff, cl-008.arff




cl-006.arff: k=3, LL = -5.876975

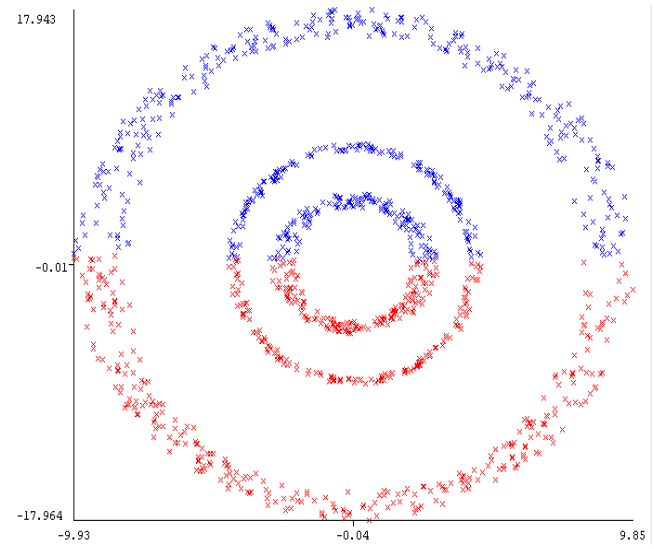



cl-007.arff: k=5, LL = -5.840204

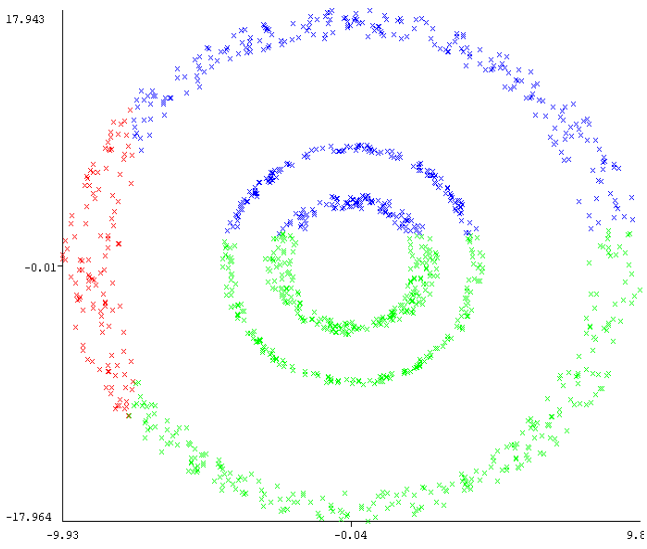



cl-008.arff: k=4, LL = -5.725845

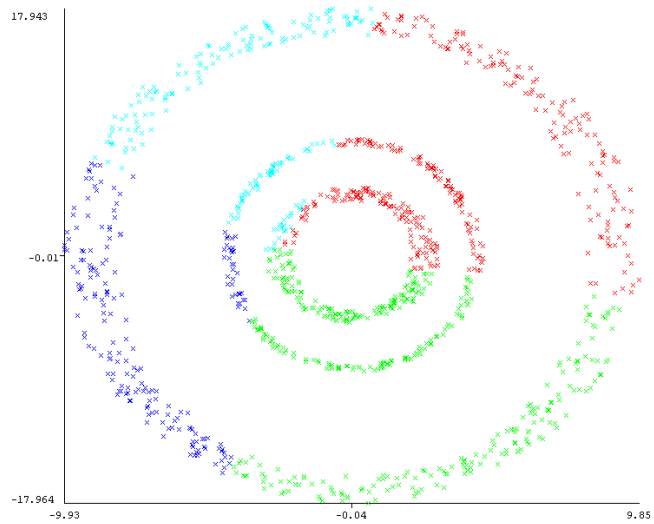
- Plik cl-009.arff




cl-009.arff: k=2, LL = -6.620152

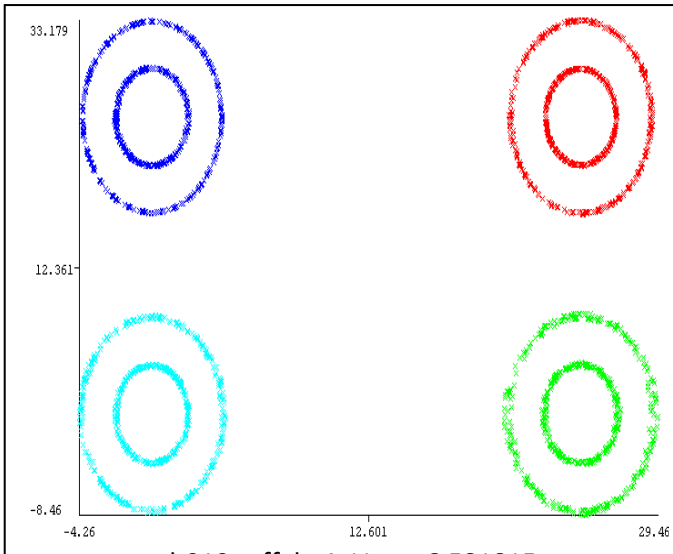



cl-009.arff: k=3, LL = -6.559911

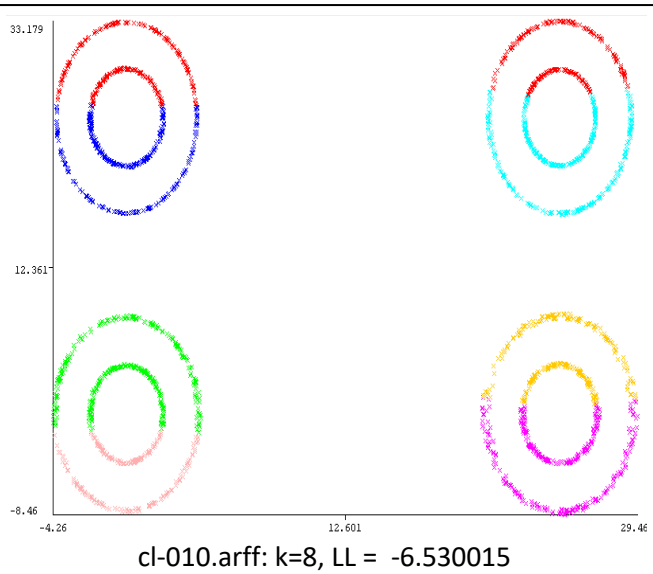



cl-009.arff: k=4, LL = -6.607904

- Plik cl-010.arff




cl-010.arff: k=4, LL = -6.581815




cl-010.arff: k=8, LL = -6.530015

- Czy da się odtworzyć kształt elips dla cl-002 i cl-003 ? Wyjaśnij.

Nie, wynika to z postaci modelu użytego w algorytmie w bibliotece Weka. Weka implementuje diagonalną macierz kowariancji, która bierze pod uwagę wyłącznie wariancje dla poszczególnych atrybutów, jest to więc model uproszczony.

- Czy da się odtworzyć kształt elips dla cl-004 i cl-005 ?

Tak, w tym przypadku udało utworzyć się kształt elips, model zadziałał poprawnie, jednak w tych przypadkach elipsy są bardziej zbliżone do koła niż w przypadkach z plików cl-002 i cl-003.

- Czy da się odtworzyć kształty skupisk dla cl-006, cl-007 i cl-008 ?

Udało się utworzyć skupiska w poprawny sposób dla cl-006, skupiska dla cl-007 i cl-008 nie są jednak utworzone w zadowalający sposób.

- Zbiór: cl-009 - przetestuj działanie algorytmu dla k=2,3,4

Najwyższy wskaźnik LL udało się uzyskać do k=3.

- Zbiór: cl-010 - przetestuj dla k=4,8

Wyższą wartość LL uzyskano dla k = 8.

### *6.6 Zrób zestawienie*

| Zbiór danych | Kmeans | DBSCAN | EM |
|---|---|---|---|
| **cl-001** | Dobry | Słaby | Słaby |
| **cl-002** | Średni | Średni | Średni |
| **cl-003** | Dobry | Dobry | Średni |
| **cl-004** | Dobry | Dobry | Dobry |
| **cl-005** | Dobry | Dobry | Dobry |
| **cl-006** | Dobry | Średni | Dobry |
| **cl-007** | Dobry | Słaby | Słaby |
| **cl-008** | Dobry | Słaby | Słaby |
| **cl-009** | Dobry | Dobry | Słaby |
| **cl-010** | Dobry | Dobry | Dobry |