Metody eksploracji danych
Laboratorium 6

Klasteryzacja

Klasteryzacja

- Celem klasteryzacja (grupowania) jest to automatyczne łączenie instancji w grupy.
- Następnie grupom można przydzielić etykiety (pełniące rolę etykiet klas)
- Zazwyczaj oczekuje się, że instancje wewnątrz grupy będą *podobne* do siebie lub *blisko położone*, natomiast nie będzie podobieństwa pomiędzy grupami (lub grupy będą *odległe*)
- W zależności od metody, liczba grup może być z góry określona lub wyznaczona przez algorytm.
- Klasteryzacja należy do problemów uczenia nienadzorowanego. Stąd nie ma jednoznacznej oceny jakości algorytmu.
  - Dla wybranych algorytmów istnieją metryki oceny (np. k-means, EM)
  - Można porównywać przypisane etykiety ze znanymi etykietami klas
  - Można analizować, jak algorytm poradzi sobie ze zbiorami o określonej topologii

Zasoby

- Podczas ćwiczeń przetwarzali będziemy sztucznie wygenerowane zbiory danych cl-xxx.arff
- Weka 3.8 (interfejs użytkownika i biblioteka weka.jar)
- Dodatkowe rozszerzenie Weka: do pobrania z http://prdownloads.sourceforge.net/weka/optics_dbScan1.0.5.zip?download
- W laboratorium jest zainstalowane oprogramowanie Eclipse Neon. Należy utworzyć projekt i dodać weka.jar oraz optics_dbScan1.0.5.jar jako biblioteki używane w projekcie
- Dokładny opis wykorzystywanych klas biblioteki znajduje się pod adresem: http://weka.sourceforge.net/doc.stable/ (ale czasem szybciej można znaleźć wpisując „Weka nazwa klasy”
- Warto też poszukać informacji pod adresem:

https://weka.wikispaces.com/Use+WEKA+in+your+Java+code

6.1 Programowe wywołanie funkcji klasteryzacji Weka

6.1.1 Załaduj plik cl-001.arff

```
DataSource source = new DataSource("cl-001.arff");
Instances data = source.getDataSet();
```

6.1.2 Przeprowadź grupowanie za pomocą metody KMeans

```
SimpleKMeans cls = new SimpleKMeans();
cls.setNumClusters(3);
cls.setPreserveInstancesOrder(true);
cls.buildClusterer(data);
```

Dodajemy atrybut wyjściowy

Zastosowany zostanie filtr Add: dodamy atrybut nominalny (wyliczeniowy) o nazwie Cluster z wartościami cluster0, cluster1, cluster2,…

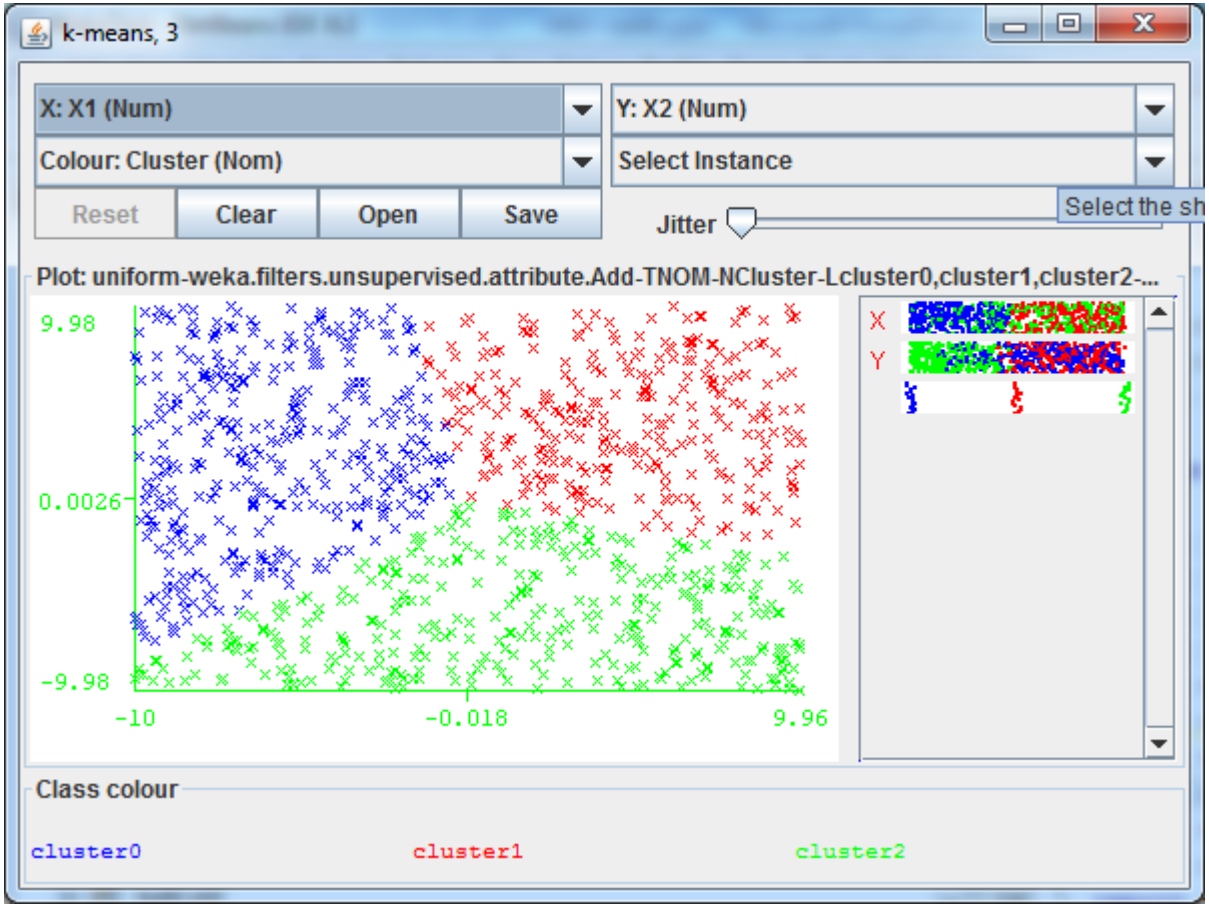
```
Add filter = new Add();
filter.setAttributeIndex("last");
int num  = cls. numberOfClusters()();
String labels = "cluster0";
for(int i=1;i<num;i++){
    labels+=", cluster";
    labels+=i;
}
filter.setNominalLabels(labels);
filter.setAttributeName("Cluster");
filter.setInputFormat(data);
Instances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

Analogicznie używa się innych filtrów Weka (linie pogrubione).

Wywołanie wizualizacji Weka

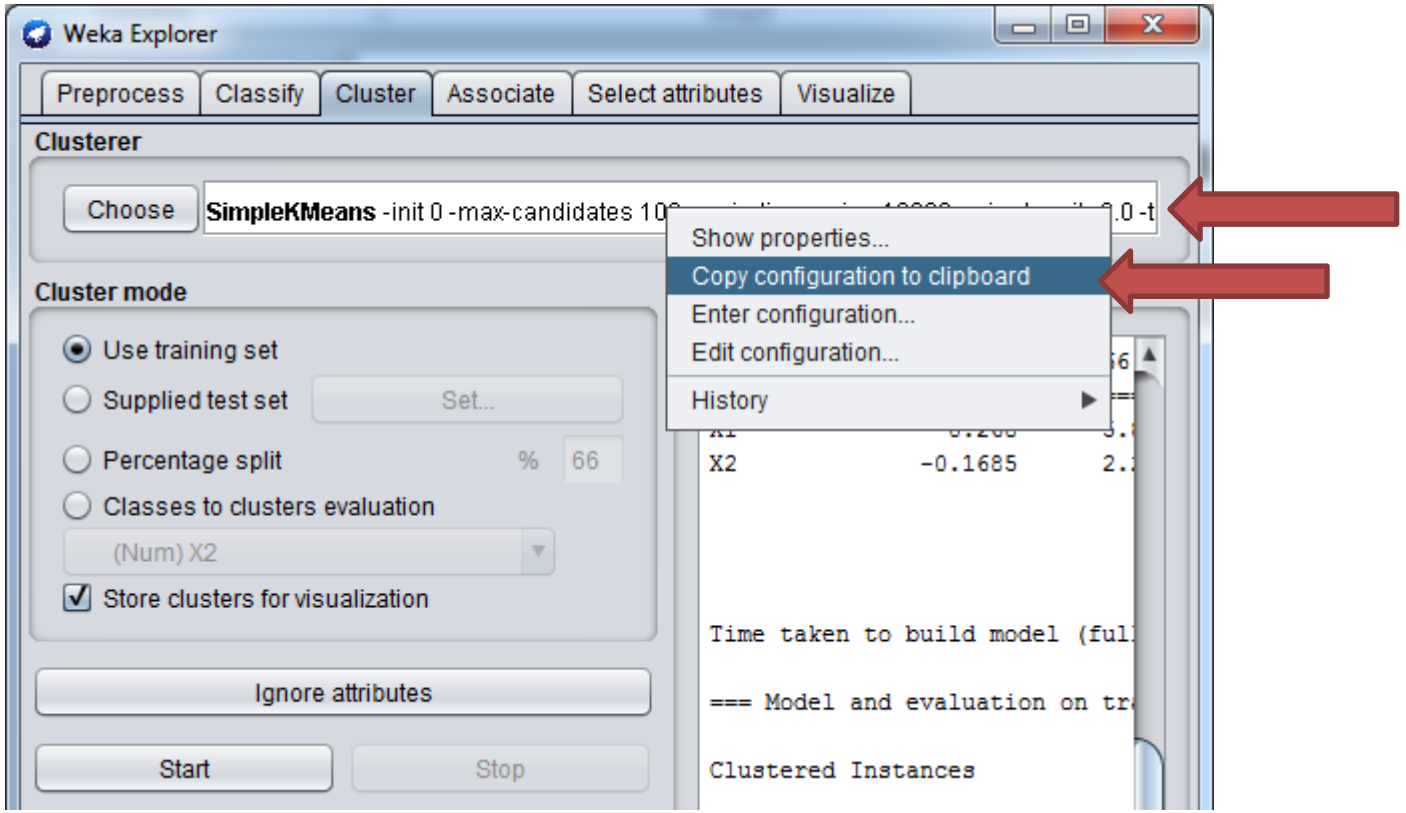
```
void visualize(Instances data, String title){
    JFrame jf = new JFrame(title);
    jf.getContentPane().setLayout(new BorderLayout());
    VisualizePanel vp = new VisualizePanel();
    vp.setShowClassPanel(true);
    jf.getContentPane().add(vp, BorderLayout.CENTER);
    jf.addWindowListener(new java.awt.event.WindowAdapter() {
        public void windowClosing(java.awt.event.WindowEvent e) {
          jf.dispose();
          System.exit(0);
        }
     });
    jf.pack();
    jf.setSize(600, 450);
    jf.setVisible(true);
    vp.setInstances(data);
}
```

Przykładowy wynik



Info: wprowadzanie parametrów

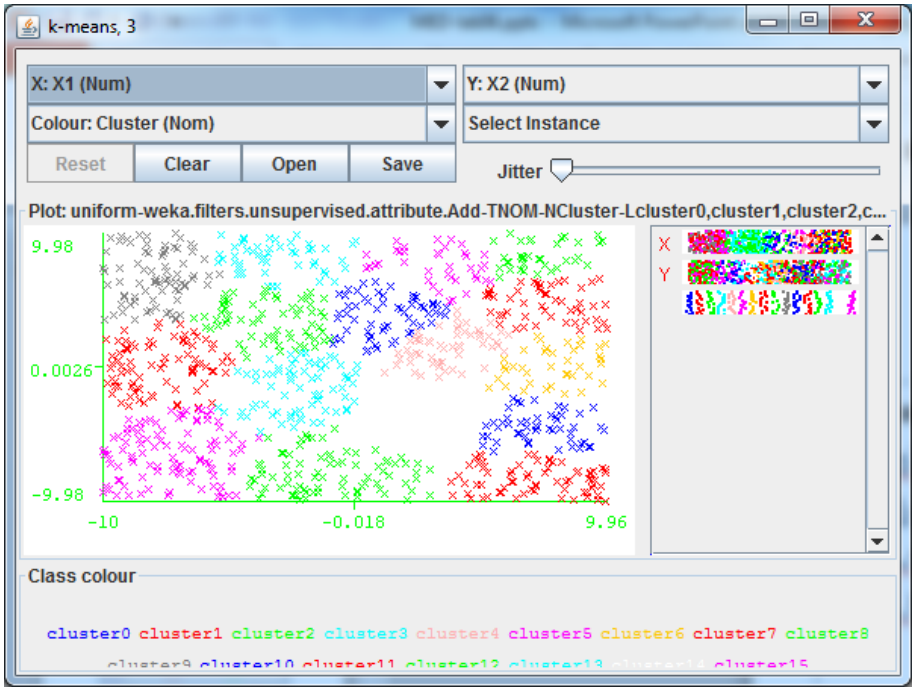
- Można jawnie, wybierając zaimplementowane metody klasy…
- Można skopiować konfigurację (prawy klawisz)



Info: wprowadzanie parametrów

- i wstawić do kodu… (tu było 16 grup)

```
SimpleKMeans cls = new SimpleKMeans();
String stringFromClipboard="-init 0 -max-candidates…";
cls.setOptions(Utils.splitOptions(stringFromClipboard));
cls.buildClusterer(data);
```



K-means

Celem algorytmu jest podział zbioru obserwacji $X = \{x_1, \cdots, x_m\}$ na k rozłącznych podzbiorów (klastów, grup) $C = C_1, \ldots, C_k$

Każda grupa C_i jest reprezentowana przez punkt środkowy (centroid) μ_i.

Funkcją celu jest:

$$J(C) = \frac{1}{k}\sum_{i=1}^{k}\sum_{x \in C_i} d(x - \mu_i)^2$$

Czyli zminimalizowanie sumy kwadratów odległości obserwacji do środków grup, do których należą.

Funkcja NP, wiele minimów lokalnych

Algorytm k-means

1. **Wylosuj** k punktów początkowych (środki grup $\mu_i, i = 1, k$) ze zbioru X
2. Dla każdej instancji $x_j \in X$ znajdź najbliższy środek μ_i i przypisz jej numer grupy i
3. Oblicz nowe środki grup $\mu_i, i = 1, k$ (jako średnie arytmetyczne współrzędnych)
4. Jeśli środki nie zmieniły się: STOP
5. Wróć do 2

Algorytm jest wrażliwy na wylosowane punkty początkowe

6.2 Przetestuj wpływ parametrów

Dodaj kod wypisujący informacje o wyznaczonych środkach klastrów i błędzie

```
Instances centroids = cls.getClusterCentroids();
for(int i=0;i<centroids.numAttributes();i++){
    System.out.print(centroids.attribute(i));
    System.out.print(",");
}

for(int i=0;i<centroids.numInstances();i++){
    System.out.println(centroids.get(i));
}
System.out.printf(Locale.US, "Error:
%f",cls.getSquaredError());
```

Przetestuj wpływ parametrów

- Dla k = 3 przetestuj kilka wartości początkowych zmieniając parametr seed generatora liczb losowych:

```
SimpleKMeans cls = new SimpleKMeans();
cls.setNumClusters(3);
cls.setSeed(10);
cls.setPreserveInstancesOrder(true);
cls.buildClusterer(data);
```

Spróbuj znaleźć 3 wartości:

- dające inne grupy (wizualizacja)
- Inne wartości funkcji błędu

Zamieść informacje o środkach grup, zwróconym błędzie i obrazki

Poeksperymentuj z parametrami k=2,3,5,9,16…

6.3 Przetwarzamy pliki za pomocą k-means

6.3.1. Plik cl-002.arff

Uruchom algorytm k-means dla k=2,3,5,8,10…

Co się dzieje z funkcją kosztu w miarę wzrostu k

6.3.2 Plik cl-003.arff

Uruchom k-means dla k=3, zapisz wyniki wizualizacji

6.3.3 Plik cl-004.arff

Uruchom k-means dla k=2,3,4, zapisz wyniki wizualizacji

6.3.4 Plik cl-005.arff

Dobierz najlepszą wartość k, zapisz wyniki wizualizacji

6.3.5 Plik cl-006.arff

Dobierz najlepszą wartość k, zapisz wyniki wizualizacji

Przetwarzamy pliki za pomocą k-means

6.3.6. Plik cl-007.arff

Dobierz najlepszą wartość k, zapisz wyniki wizualizacji

6.3.7. Plik cl-008.arff

Dobierz najlepszą wartość k, zapisz wyniki wizualizacji

6.3.8. Plik cl-009.arff

Zapisz wyniki wizualizacji dla k=2,3

6.3.9. Plik cl-010.arff

Dobierz najlepszą wartość k, zapisz wyniki wizualizacji

6.4 Algorytm DBSCAN

DBSCAN: **Density-based spatial clustering of applications with noise.**

Algorytm łączy w klastry obserwacje leżące blisko siebie (wg zasady sąsiad mojego sąsiada należy do tej samej grupy, co ja).

Algorytm ignoruje pojedyncze punkty lub niewielkie skupiska

```
DBSCAN cls = new DBSCAN();
cls.setMinPoints(3);
cls.setEpsilon(???);
cls.buildClusterer(data);
```

Dwa paramtery

- epsilon - jeśli dla dwóch punktów odległość $d(x_1, x_2) < epsilon$, to należą do tej samej grupy
- min_points – ignorowane są grupy $|C_i| <= \min\_points$ (zawierające mniej niż $\min\_points$ obserwacji)

Niestety kod biblioteki (rozszerzenia) jest źle napisany, ale można go użyć do pokazania zasady działanie

Przetwarzamy pliki…

6.4.1 cl-0002.arff zapisz wizualizację

6.4.2 cl-0001.arff (Co się dzieje i dlaczego? Ile klastrów zostało wykrytych)

6.4.3 cl-0003.arff Dlaczego pojawiają się wyjątki?

Popraw kod dodający numer klastra:

```
int idx = newData.numAttributes()-1;
 for(int i=0;i<newData.numInstances();i++){
    try{
       int val = cls.clusterInstance(data.get(i));
       if(val==-1)continue;
       newData.get(i).setValue(idx, val);
    }
    catch(Exception e){}
 }
```

Poeksperymentuj z parametrem epsilon (wartości <=0.1) zapisz wizualizację

Przetwarzamy pliki…

6.4.4 cl-0004.arff zapisz wizualizację

6.4.5 cl-0005.arff dobierz wartość epsilon, zapisz najlepszą wizualizację

6.4.6 cl-0006.arff dobierz wartość epsilon, zapisz najlepszą wizualizację

6.4.7 cl-0007.arff i cl-0008.arff powinny dać analogiczne wyniki, dlaczego?

6.4.8 cl-0009.arff dobierz wartość epsilon, zapisz najlepszą wizualizację

6.4.9 cl-0010.arff dobierz wartość epsilon, zapisz najlepszą wizualizację

Expectation–maximization EM

- Algorytm zakłada, że obserwacje są wygenerowane na podstawie k rozkładów gaussowskich $N(\mu_i, \sigma_i^2)$ o różnych punktach środkowych μ_i oraz wariancjach σ_i^2.
- Kryterium jakości klasteryzacji jest iloczyn prawdopodobieństw, że obserwacja x_j należy do i-tej grupy. Ale ponieważ iloczyn małych wartości byłby niestabilny numerycznie – obliczana jest suma logarytmów (log likehood).
- Aby obliczyć funkcję kosztu (log-likehood): należy iterować po instancjach i zsumować
  cls.logDensityForInstance(data.get(i));
- EM zaimplementowany w Weka sam może dobrać liczbę klastrów (testując kolejne wartości k i obliczając log-likehood). Trwa to długo, więc lepiej ustawić wartość ręcznie.

6.5. Przetwarzamy pliki

6.5.1 Dodaj kod obliczający log likehood dla oceny jakości

```
for(int i=0;i<data.numInstances();i++){
        double density = cls.logDensityForInstance(data.get(i));
        logLikehood+=density;
}
System.out.printf(Locale.US, "LL: %f",logLikehood/data.numInstances());
```

6.5.2 Zadeklaruj obiekt EM i ustaw k

```
EM cls = new EM();
cls.setNumClusters(k);
cls.setSeed(10);
cls.buildClusterer(data);
```

Przetwarzamy pliki

- 6.5.3 Uruchom EM dla kolejnych plików cl-xxx.arff, zapisuj wizualizacje i log-likehood
- Odpowiedz na pytania:
  - 6.5.3.1 Czy da się odtworzyć kształt elips dla cl-002 i cl-003? Wyjaśnij.
  - 6.5.3.2 Czy da się odtworzyć kształt elips dla cl-004 i cl-005?
  - 6.5.3.3 Czy da się odtworzyć kształty skupisk dla cl-006, cl-007 i cl-008?
  - 6.5.3.4 Zbiór: cl-009 - przetestuj działanie algorytmu dla k=2,3,4
  - 6.5.3.5 Zbiór: cl-010 - przetestuj dla k=4,8

6.6 Zrób zestawienie

Zbiór danych	K-means	DBSCAN	EM
Cl-001			
Cl-002			
itd			
CL-XXX	Dobry	Średni	Średni

Oceń jakość klasteryzacji dla konkretnych zbiorów używając terminów: Dobry, Średni, Słaby