1) Temat projektu

Stworzenie wybraną metodą **klasyfikatora** dla danych parametrycznych pobranych z rzeczywistego modelu silnika indukcyjnego w oparciu o wyznaczone klastry obliczone z pomocą algorytmu grupującego *k-means*.

2) Charakter danych

Otrzymane pliki:

* ***training\_5th\_order\_tolerances10\_exhaustive.txt***

o rozmiarze: 180x56,

180 wektorów danych posiadających 55 atrybutów oraz kategorię,

* ***testing\_5th\_order\_tolerances10\_exhaustive.txt***

o rozmiarze: 180x56,

180 wektorów danych posiadających 55 atrybutów oraz kategorię,

* ***rs\_train5\_noheaders.txt***

o rozmiarze: 70x55,

70 wektorów danych posiadających 55 atrybutów,

* ***rs\_test5\_noheaders.txt***

o rozmiarze: 70x55,

70 wektorów danych posiadających 55 atrybutów.

3) Analiza danych

Z pliku ***training\_5th\_order\_tolerances10\_exhaustive.txt*** została wyekstrahowana ostatnia kolumna zawierająca kategorię. Spośród **180** elementów aż **105** jest unikatowych co do wartości, a ich zakres jest od **2.6e-9** do **1900**.

Z pliku ***testing\_5th\_order\_tolerances10\_exhaustive.txt*** została wyekstrahowana ostatnia kolumna zawierająca kategorię. Spośród **180** elementów aż **107** jest unikatowych co do wartości, a ich zakres jest od **1.3e-9** do **1860**.

Dla sprawdzenia wartości kategorii z omówionych powyżej plików stworzono jednolity wspólny wektor o długości **360** (pobrane 180 elementów z pierwszego pliku oraz 180 elementów z drugiego pliku) i sprawdzono, ile klas jest przecinających się tzn. o dokładnie tych samych wartościach. Otrzymano wynik na poziomie: **6**, zawierający elementy: **1.3e-8**, **1.6e-8**, **1.9e-8**, **4.9e-8**, **5.1e-8** i **1000**.

Wykorzystanie algorytmu grupującego *k-means* zostanie oparte na znormalizowanej wartości kategorii dla obu ww. plików, tzn. kategorie będą przyjmować wartości od **1** do **n.** Określenie odpowiednich przedziałów umożliwi uzyskanie zakresu wartości, dla którego będzie można przeprowadzić klasteryzację danych i porównanie grup z kategoriami danych.

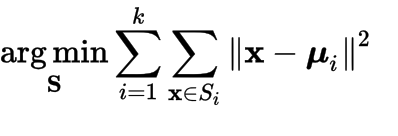
Pliki: ***rs\_train5\_noheaders.txt*** oraz ***rs\_test5\_noheaders.txt***, które nie posiadają etykiet danych, mogą zostać wykorzystane jako dodatkowe dane do nienadzorowanej metody uczenia typu *k-means* i grupowania na reprezentatywnym zbiorze danych. W tej sytuacji rozmiar danych będzie równy: **320** wektorów danych (bez pliku: ***testing\_5th\_order\_tolerances10\_exhaustive.txt***).

4) Część pierwsza

Algorytm grupujący *k-means* wykorzystuje następujące parametry:

* liczba klastrów **k** - wartość zmienna od **1** (pierwszy skrajny przypadek) do **360** (drugi skrajny przypadek) - praktycznie **10 - 200**,
* wybór odległości - *sqeuclidean*, *cityblock*, *cosine*, *correlation*, *hamming*, razem **5** opcji,
* maksymalna liczba iteracji **max** - domyślnie **100**,
* liczba powtórzeń przy nowych warunkach początkowych **replicates** - **100**.

Posiadając zbiór obserwacji (x1, x2, ..., x*n*), gdzie każda obserwacja jest wektorem rzeczywistym o liczbie wymiarów wynoszącej d, klastrowanie k-średnich stara się podzielić owe n obserwacji na *k* (≤ *n*) zbiorów S = {*S*1, *S*2, ..., *Sk*} tak, by zminimalizować wewnątrzklastrową sumę kwadratów (WCSS, within-cluster sum of squares), czyli sumę funkcji odległości każdego z punktów klastra do centrum K. Inaczej ujmując, celem algorytmu jest znalezienie :



Funkcja oceny poszukiwanego globalnego optymalnego rozwiązania obejmuje minimalną liczbę klastrów **k** przy minimalizacji wzajemnego dystansu od grup (np. narzędzie Matlab umożliwia kontrolować tę wartość na wyjściu funkcji *kmeans*).

Przykładowe rozwiązania problemu będą porównywane na bieżąco z kategoriami w pliku uczącym i wyznaczane prawdopodobieństwa przynależenie do nowego klastra. Pobocznym celem jest zoptymalizowanie tych wartości do maksymalnie **100 %** - interesuje nas, żeby wybrany wektor danych był przydzielony tylko do jednego klastra, a nie do kilku-kilkunastu, gdzie prawdopodobieństwo jest dużo niższe.

Metoda dochodzenia do ekstremum dla funkcji oceny będzie obejmować przyrostowe zmiany ww. parametrów (pełny przegląd), jak również wykorzystanie obliczanych prawdopodobieństw (wpływających na umiejętną zmianę kroku).

Klastrowanie k-średnich jest nienadzorowanym algorytmem uczenia, który stara się pogrupować dane w oparciu o ich podobieństwo. Nienadzorowane uczenie niesie ze sobą fakt, że rezultat jest trudny do przewidzenia i algorytm iteracja po iteracji stara się odnaleźć wzorce pośród danych. W klastrowaniu k-średnich potrzebujemy określić:

• liczbę klastrów, w które chcemy pogrupować dane, oraz

• liczbę iteracji algorytmu.

Algorytm losowo przypisuje każdą obserwację do klastra i znajduje centroid każdego z klastrów. Następnie, algorytm iteruje przez dwa kroki :

• Przenieś przykłady do klastra, którego centroid znajduje się najbliżej, a następnie

• Oblicz nowy centroid dla każdego z klastrów.

Powyższe kroki powtarzane są aż wariancja wewnątrz klastra osiągnie minimalną wartość. Wariancja ta obliczana jest jako suma odległości euklidesowych pomiędzy punktami danych i odpowiednimi centroidami klastrów.

5) Środowisko analityczne

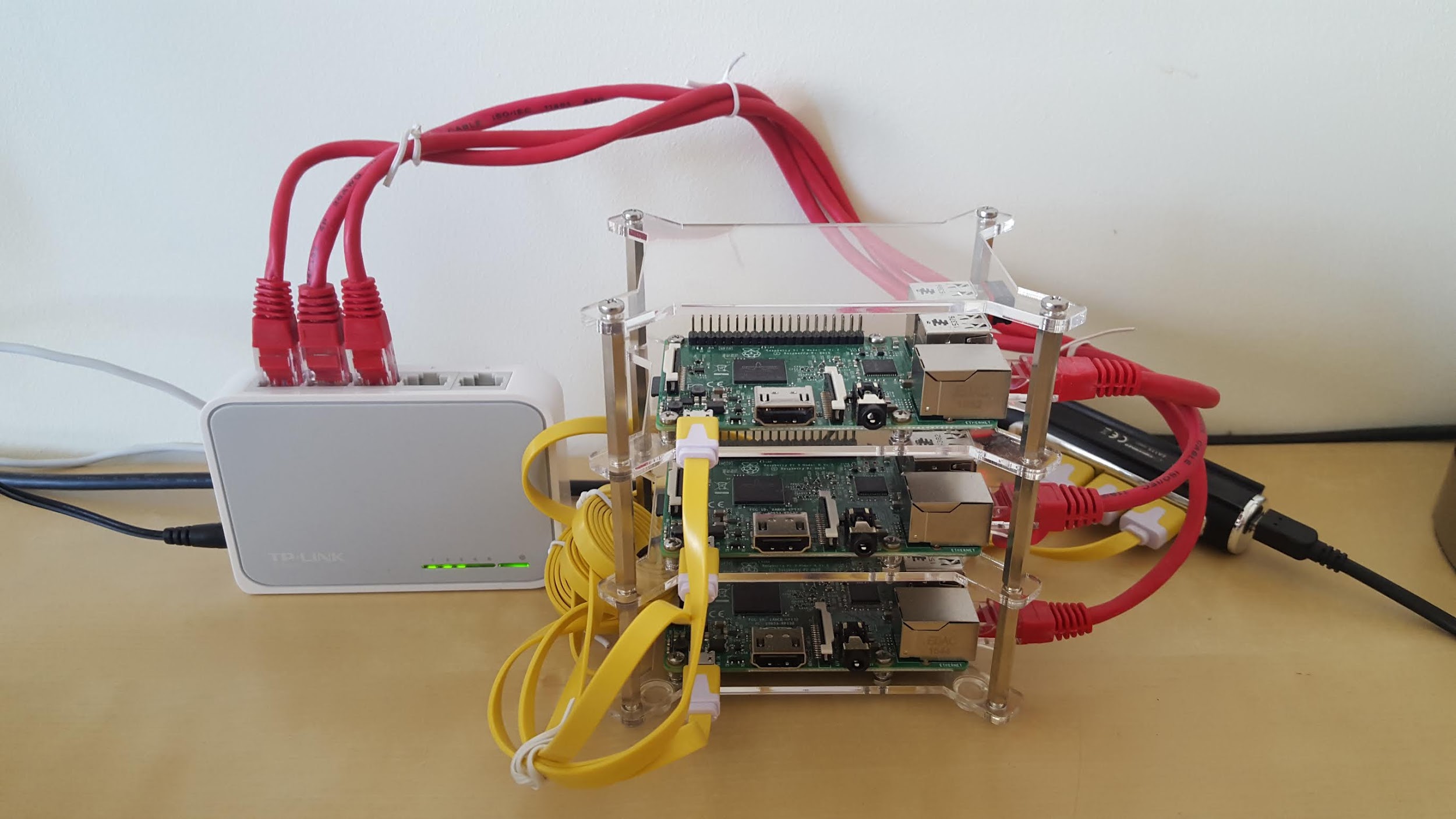
Implementacja algorytmu k-means będzie miała miejsce w środowisku rozproszonym Apache Spark. Jest to komponent ekosystemu Hadoop służący do przetwarzania i analizowania dużych zbiorów danych zarówno w trybie wsadowym, jak i czasu rzeczywistego. Charakteryzuje się bardzo dużą szybkością działania ze względu na operowanie na danych w pamięci operacyjnej maszyn wchodzących w skład klastra.

Apache Spark zawiera bibliotekę MLlib składającą się z implementacji wielu popularnych algorytmów uczenia maszynowego, w tym m.in. k-means.

Na każdej maszynie klastra, zainstalowany został system Linux Raspian oparty o dystrybucję Debian. Maszyny zostały połączone w sieć lokalną z wykorzystaniem przełącznika oraz dodatkowo każda z maszyn ma połączenie z Internetem dzięki wbudowanej karcie bezprzewodowej. W celu zwiększenia przepustowości klastra w trakcie obliczeń maszyny komunikują się przez sieć wewnętrzną.

Na każdej z maszyn zainstalowana została dystrybucja Hadoop 2.7.3 wraz z frameworkiem Apache Spark w wersji 2.0.3. Językiem implementacji będzie Scala.

Na zdjęciu poniżej pokazana jest budowa klastra.



Na tym etapie projektu będziemy musieli uwzględnić odpowiedzi na pytania:

a) Ile klastrów wybrać ?

b) Czy maksymalizować miarę dopasowania R-kwadrat czy minimalizować poziom entropii ? Czy zastosować miarę szerokości sylwetki ?

c) Czy normalizować dane wejściowe ?

6) Część druga

Po dokonaniu klasteryzacji danych określimy, jakie etykiety klas (zadane początkowo) z jakim rozkładem należą do poszczególnych klastrów. Rozkład ten określimy na podstawie stosunku ilości rekordów danej klasy w danym klastrze do łącznej ilość rekordów danej klasy. Następnie określimy prawdopodobieństwo wystąpienia poszczególnych klas w danym klastrze poprzez przeskalowanie rozkładu tych klas oraz proporcji wystąpień rekordów z poszczególnych klas w klastrze względem siebie.

*Przykład*:

* Klasa A ma łącznie 10 rekordów
* Klasa B ma łącznie 1 rekord
* Klaster I ma łącznie 2 rekordy: jeden klasy A oraz jeden klasy B
* Rozkład klasy A dla klastra I wynosi 1 / 10 = 0.1
* Rozkład klasy B dla klastra I wynosi 1 / 1 = 1
* Przeskalowane prawdopodobieństwo w klastrze I dla klasy A wynosi 0.1 / (0.1 + 1) = 9%
* Przeskalowane prawdopodobieństwo w klastrze I dla klasy B wynosi 1 / (0.1 + 1) = 91%
* Interpretacja: na 91% rekord w klastrze I powinien należeć do klasy B

Kolejnym krokiem będzie zbudowanie modelu klasyfikacyjnego opartego na algorytmie drzewa decyzyjnego / naiwnego bayesa z wektorem zmiennych zależnych postaci etykiet utworzonych w poprzedniej części projektu klastrów.

Finalnym analitycznym etapem będzie porównanie predykcji przynależności do danego klastra a tym samym z klasą, która w danym klastrze jest najbardziej prawdopodobna z klasą którą mieliśmy w etykiecie danych wejściowych.

7) Realizacja

Implementacja algorytmy k-means została zrealizowana w środowisku Apache Spark jak również część kodu została napisała w czystym Pythonie. Pierwszym zagadnieniem było ustalenie ilości klastrów. Miarą rozwiązania jest wartość WCSS (within-cluster sum of squares), czyli suma funkcji odległości każdego z punktów klastra do centrum K.

Testowany zakres ilość klastrów wyniósł 10 - 200:

for i in range(10, 200):

kmeans = KMeans(n\_clusters = i, init = 'k-means++', random\_state = 42)

kmeans.fit(X)

wcss.append(kmeans.inertia\_)

kmeans.fit(X\_normalized)

wcss\_normalized.append(kmeans.inertia\_)

Do podjęcia decyzji o ilości klastrów wykorzystaliśmy metodę Elbow, czyli graficzna metoda stwierdzająca przy jakiej ilości klastrów błąd jest najmniejszy, a dalsze zwiększanie ilości tych klastrów powoduje już znacznie mniejszy spadek błędu.

W przypadku danych nie znormalizowanych, wartość ta oscylowała w granicach 20:

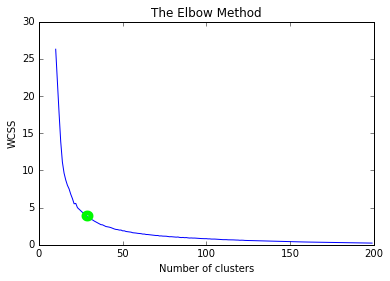
plt.plot(range(10, 200), wcss)

plt.title('The Elbow Method')

plt.xlabel('Number of clusters')

plt.ylabel('WCSS')

plt.show()



Natomiast w zbiorze danych powstałym przez połączenie zbioru trenującego i testowego (w przypadku k-means, mamy uczenie bez nadzoru także nie potrzebujemy danych testowych) znalazło się 44 przypadków klas (dane wejściowe zawierały etykiety) z liczebnością co najmniej dwa rekordy. Sugerowało to liczebność klastrów co najmniej na tym samym poziomie.

Nie wiemy w jaki sposób powstały przypisania klas do poszczególnych rekordów, nie mniej spróbowaliśmy znormalizować dane wejściowe.

Algorytm K-means jest szczególnie wrażliwy na nieznormalizowane dane ponieważ w naszej implementacji używaliśmy miary odległości Euklidesowej i wszystkie zmienne powinny mieć podobny zakres zmienności.

Najczęściej spotykanym sposobem normalizacji jest tzw. *standaryzacja Z*, którą można wyrazić poniższym wzorem:

Zrzut ekranu 2017-01-23 o 17.43.25.png

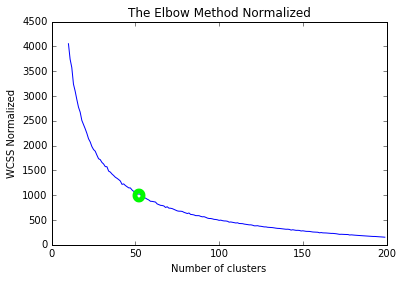
gdzie:

*x* - zmienna niestandaryzowana

*μ* - średnia z populacji.

*σ* - odchylenie standardowe populacji

Po znormalizowaniu, ponownie korzystając z metody Elbow, okazało się, że optimum znajduje się w okolicach 50. Zgadzało się to z naszymi założeniami więc przyjęliśmy wartość 50 w dalszej implementacji.



Wyniki przedstawia tabela poniżej:

+----------+-----+

|prediction|count|

+----------+-----+

| 14| 17|

| 24| 16|

| 47| 15|

| 32| 15|

| 37| 14|

| 20| 13|

| 42| 13|

| 0| 13|

| 7| 12|

| 35| 11|

| 28| 11|

| 39| 11|

| 9| 11|

| 33| 10|

| 11| 10|

| 43| 10|

| 18| 10|

| 8| 9|

| 30| 9|

| 12| 8|

| 27| 8|

| 49| 8|

| 48| 8|

| 3| 7|

| 1| 7|

| 29| 7|

| 41| 6|

| 19| 6|

| 40| 6|

| 22| 5|

| 23| 5|

| 4| 5|

| 6| 5|

| 16| 4|

| 13| 4|

| 15| 4|

| 10| 4|

| 36| 3|

| 45| 3|

| 31| 3|

| 26| 2|

| 44| 2|

| 5| 2|

| 46| 2|

| **34**| **1**|

| **17**| **1**|

| **25**| **1**|

| **2**| **1**|

| **38**| **1**|

| **21**| **1**|

+----------+-----+

Okazało się, że po znormalizowaniu dokładnie 44 klastry zawierały więcej niż jeden rekord. Utwierdziło nas to w przekonaniu, że wybrana ilość klastrów jest słuszna.

Zapewne dalsze zmniejszanie ilości klastrów (do 44 włącznie) mogłoby doprowadzić nas do jeszcze lepszych rezultatów jednak zauważyliśmy, że już przy ilości 50 klastrów, odpowiednio dobierając centroidy startowego (modyfikując wartość ziarna) mogliśmy doprowadzić do stanu gdy tylko 4 klastry miały liczność 1.