Praca domowa nr 2

Paweł Koźmiński

09.05.2019

1. Wprowadzenie

Tematem drugiej pracy domowej była analiza podziału zbioru danych na skupienia (ang. *data clustering*). Jest to metoda klasyfikacji bez nadzoru. Polega na automatycznym pogrupowaniu zbioru danych na rozłączne klasy pod względem pewnego "podobieństwa" punktów. Podobne sobie punkty tworzą jedno określone skupienie.

Algorytmy klasteryzacji

Problem grupowania danych może być rozwiązany za pomocą wielu algorytmów. Za Wikipedią, dzielą się one na:

- algorytmy hierarchiczne
- grupy metod k-średnich
- metody rozmytej analizy skupień.

Jaki jest cel zadania?

Zadanie 2. pracy domowej tak naprawdę składało się z wielu poleceń. Pierwszym, być może najciekawszym krokiem było samodzielne stworzenie funkcji dokonującej podziały zbioru danych na skupienia. Następnie, korzystając z bibliotek `stats`, `genie` oraz innej, wybranej przez siebie, należało dokonać porównania skuteczności różnych funkcji dokonujących grupowania na kilkudziesięciu zbiorach testowych. Oprócz tego, należało stworzyć kilka własnych zbiorów danych do testowania owych algorytmów. Opis owych zbiorów danych znajduje się w załączonym pliku *testy*. Ostatecznie, liczba wszystkich zbiorów testowych wyniosła 46, zawierają one współrzędne punktów w $\mathbb{R}^2$, $\mathbb{R}^3$, a także w $\mathbb{R}^4$.

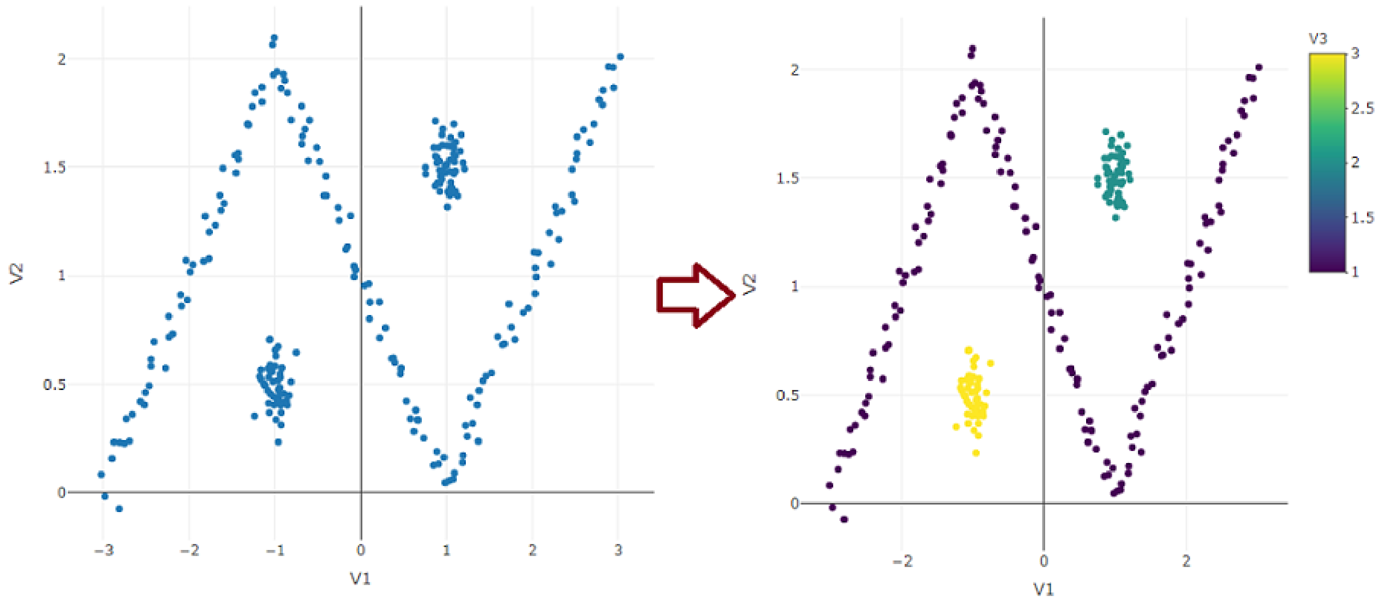


Figure 1: Ilustracja działania analizy skupień

2. Wybrane algorytmy analizy skupień

Pakiet `stats`

Pakiet `stats` zawiera rodzinę hierarchicznych algorytmów wbudowanych w funkcji `hclust()`.
Metody do wyboru to: *Single*, *Complete*, dwie *metody Warda*, *metoda McQuitty'ego*, *median* oraz *centroidów*. W celu porównaniu, wypróbowane zostały wszystkie z nich.

Algorytm `genie`

Jak można przeczytać w opisie funkcji `hclust2()` pakietu `genie`, jest to szybki algorytm hierarchiczny. Owa funkcja zawiera parametr *thresholdGini* - liczbę z przedziału [0,1], próg indeksu *Gini*, którego przetestowane zostały różne wartości.

Pakiet `cluster`

Dowolnie wybraną bilioteką z archiwum *CRAN* została `cluster` wraz z funkcją `agnes()` oferującą kilka hierarchicznych funkcji. Próbując wszystkie z dostępnych metod na kilku zbiorach benchmarkowych (z katalogu *graves*) otrzymałem następujące wyniki:

	Index mean
Single	0.6540335
Average	0.5674242
Complete	0.4882164
Ward	0.5493227
Weighted	0.4639858
GAverage	0.5352033

Ponadto, metoda *Single* charakteryzowała się jednym z krótszych czasów działania. Mimo to warto nadmienić, że zazwyczaj działała ona dłużej niż wszystkie 7 metod funkcji `hclust()`.

Opisane dotychczas algorytmy hierarchiczne zostały przetworzone za pomocą funkcji `cutree()` w celu otrzymania właściwej listy etykiet.

Algorytm spektralny

Funkcja stworzona samodzielnie jest implementacją algorytmu spektralnego, korzystającego choćby z M najbliższych sąsiadów czy algorytmu k-średnich.

To właśnie parametr M mógł być modyfikowany, co na potrzeby eksperymentu czyniłem.
Początkowo odnosiłem wrażenie, że im większą przyjmiemy wartość M, tym wynik powinien być wyższy. Z tego powodu przyjąłem 4 wartości M, zależne od liczby punktów zbioru oraz liczby klas. Kolumna `ownq1` przyjmuje za M 25% liczby obserwacji, `own2q` 50%, `own3q` 75%, a `own` - liczbę: $\frac{liczba\ obserwacji}{liczba\ oczekiwanych\ grup}$.
Po kilku testach zrozumiałem, że M powinno oscylować wokół kilku, kilkunastu. Z tego powodu przyjąłem wartości M: 5, 10, 12, 15. Zdecydowałem się jednak nie porzucać otrzymanych wyników i pozostały one w wynikach testów jako materiał badawczy.

3. Sprawdzenie skuteczności

Indeksy

Skuteczność poszczególnych algorytmów sprawdzona zowstała za pomocą dwóch narzędzi: indeksu Fowlkesa-Mallowsa(FM) oraz skorygowanego indeksu Randa(AR). Odpowiednie funkcje znajdują się w bibliotekach `dendextend` oraz `mclust`. Oba indeksy zwracają wartości nie większe od 1, gdzie 1 oznacza w pełni poprawny podział zbioru na klastery, a im dalej od tej wartości, tym gorszy był wynik działania algorytmu.

Standaryzacja

Dodatkowym czynikiem, mającym wpływ na skuteczność algorytmów analizy skupień, jest standaryzacja danych. Odpowiada za nią funkcja `scale()` w bazowym *R*.

4. Otrzymane wyniki testów na zbiorach

Standaryzacja

Sprawdzimy najpierw, jak na wyniki osiągi sprawdzanych funkcji wpłynęła wspomniana standaryzacja.

	Bez standaryzacji	Po standaryzacji
Średni indeks	0.6475576	0.6270796

Jak widać, wartości indeksów wyliczanych nazbiorach bez standaryzacji są minimalnie wyższe, zatem wpływa ona negatywnie na skuteczność algorytmów. Sprawdźmy zatem, w ilu przypadkach, uwzględniając jedynie sytuacje, w których otrzymane wyniki były różne, brak standaryzacji korzystnie wpłynął na wynik badania. (*Figure 2*)

W związku z osiągniętymi rezultatami, w dalszej części będziemy zajmować się jedynie indeksami otrzymanymi na zbiorach nieskalowanych.

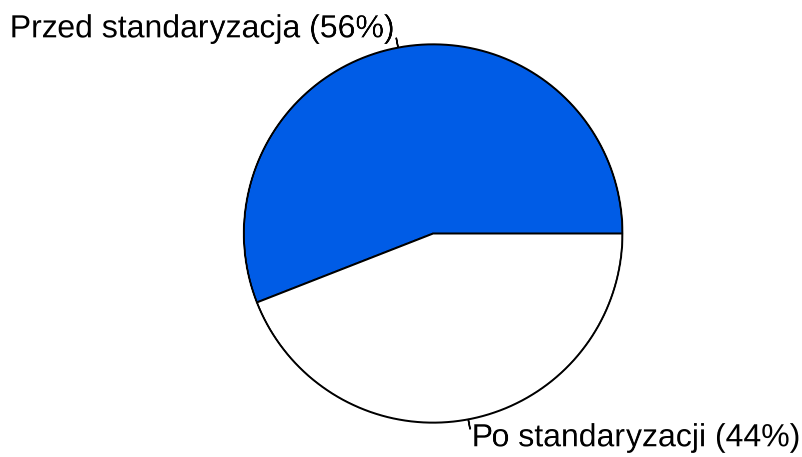
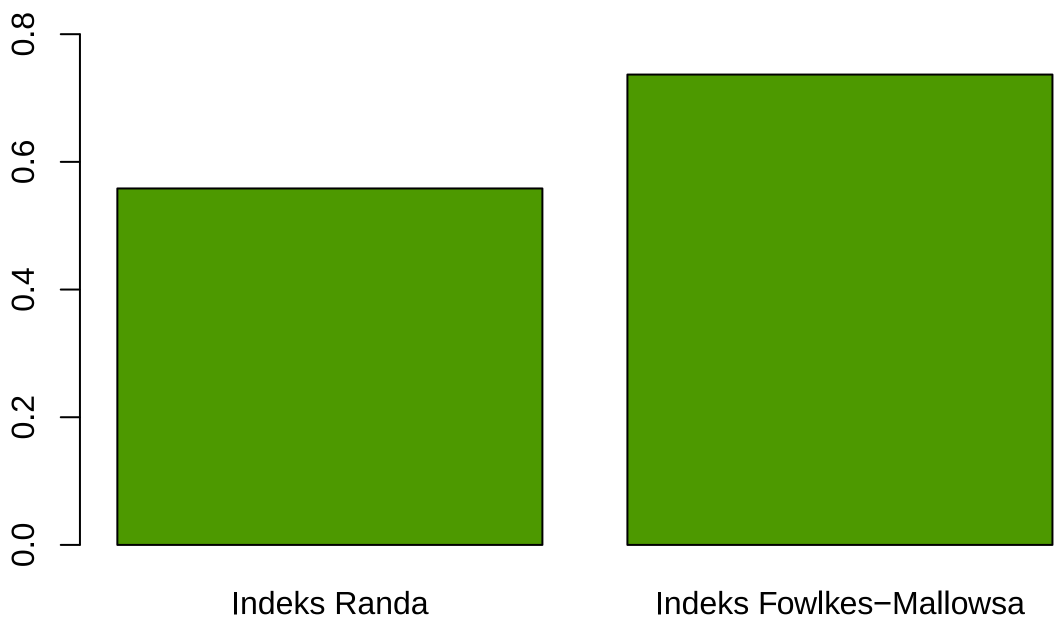


Figure 2: Jak często wyniki przed standaryzacją były wyższe niz po?

Działanie indeksów

Sprawdźmy zatem, który z indeksów sprawdzających skuteczność klasteryzacji był dla zbiorów bardziej surowy, obliczając ich średnie wartości na wszystkich zbiorach.

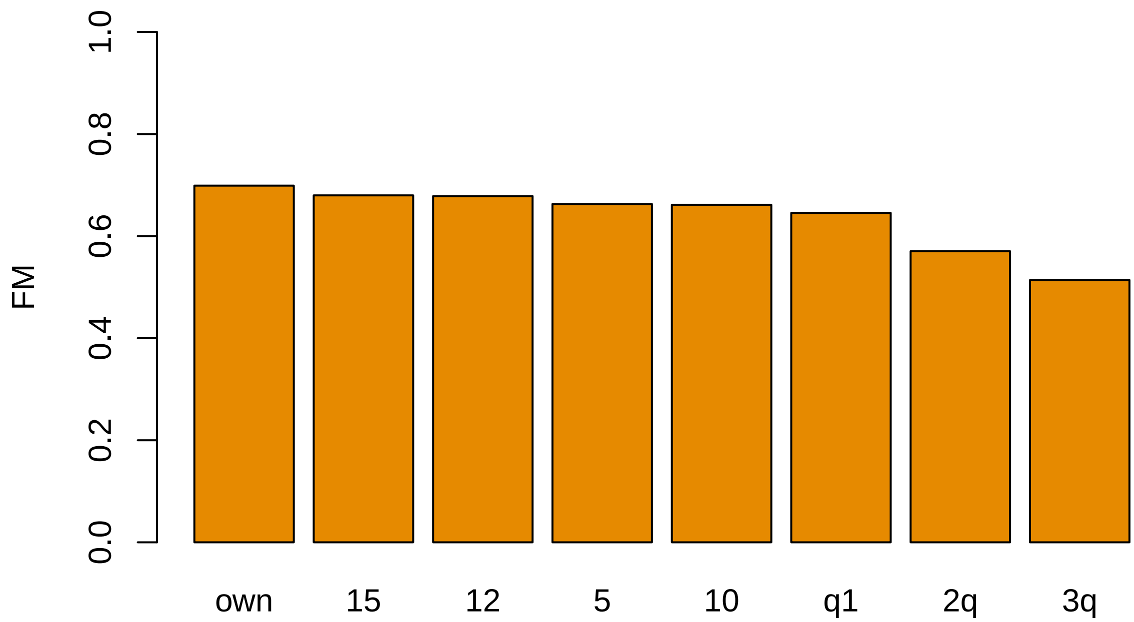


A czy wyniki działania obu indeksów były zbieżne? Sprawdzimy to na podstawie średnich wyników indeksów dla każdejz funkcji grupujących, badając następnie ich korelację:

Wartość korelacji
0.9687587

Jak widzimy, wartość współczynnika korelacji między średnimi indeksów jest wysoka, zatem oba indeksy bardzo podobnie oceniały skuteczność algorytmów. Ponieważ zachodzi taka zależność, weźmy pod uwagę “łagodniejszy” z nich: indeks Fowlkesa-Mallowsa.

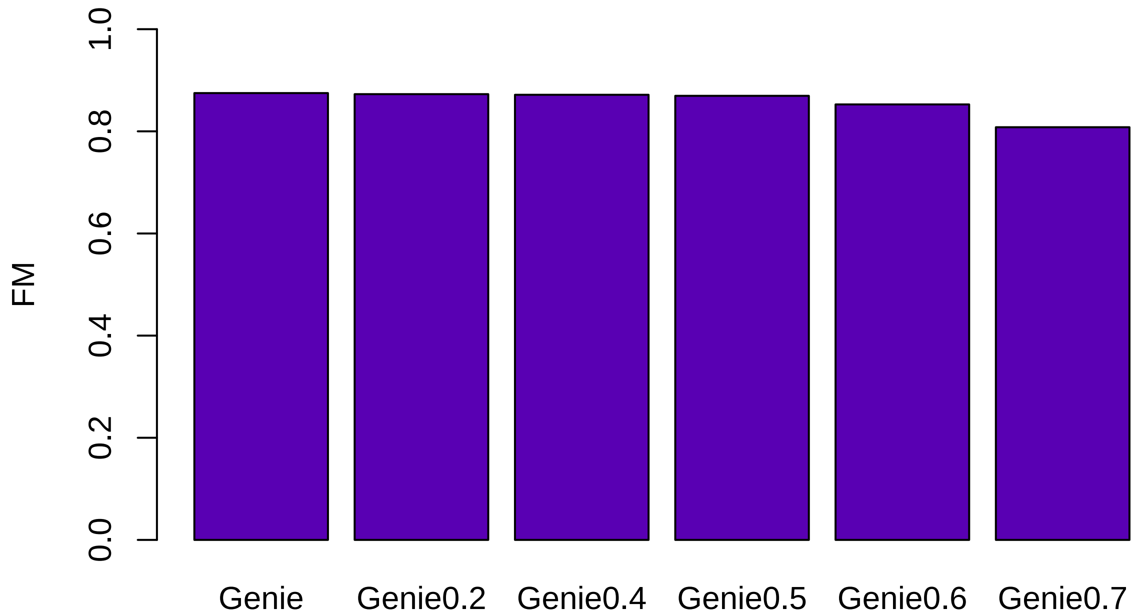
Sprawdzenie wyników algorytmów spektralnych



Wynik tej analizy może być zaskakujący. Największą skutecznością wykazała się funkcja, której wartość M była modyfikowana w zależności od liczby elementów zbioru. Przypomnijmy, $M = \frac{liczba\ obserwacji}{liczba\ oczekiwanych\ grup}$. Dostrzegalny może być nieznaczny wzrost skuteczności funkcji między M równym 5 i 10 a 12 i 15. Najgorzej - bez zaskoczenia - wypadła wersja `own3q`.

Analiza algorytmów biblioteki `genie`

Funkcja `hclust()` z biblioteki `genie` została testowana dla różnych wartości *threshold.* Sprawdźmy, który z nich najlepiej sobie poradził z powierzonym mu zadaniem:

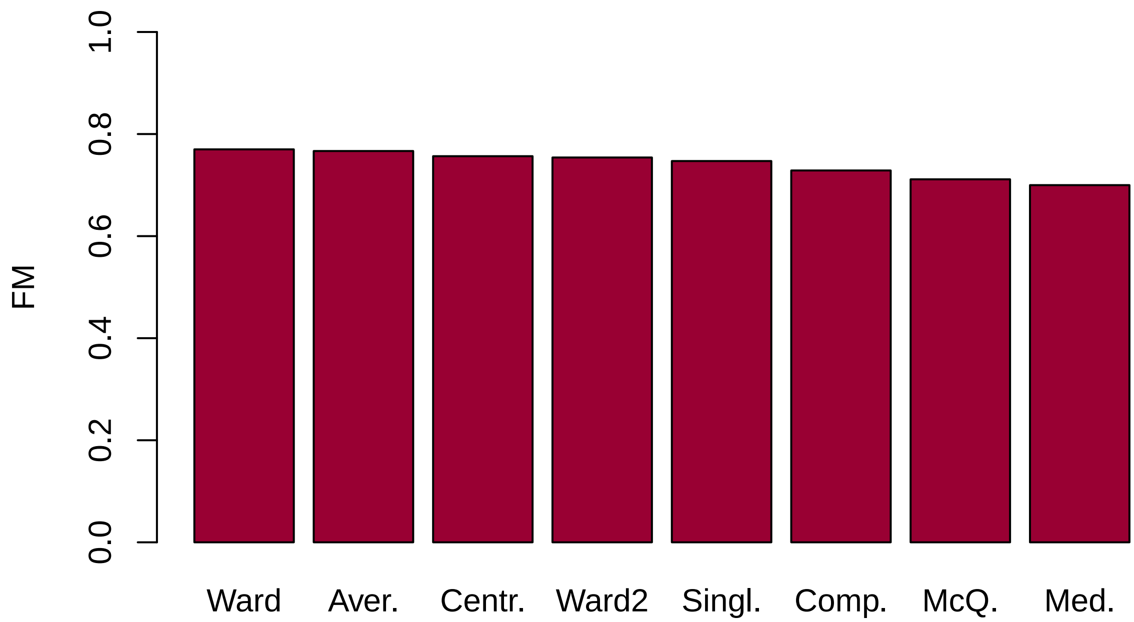


Gwoli ścisłości - wartości odpowiadające słupkom zostały posortowane malejąco:

Genie	Genie0.2	Genie0.4	Genie0.5	Genie0.6	Genie0.7
0.8748965	0.87281	0.8715365	0.869512	0.8527128	0.8080661

Domyślna wartość wspomnianego parametru - 0.3 - okazała się być najlepszą. Im bardziej *threshold* się od niej różnił, tym słabsze wychodziły wyniki podziału zbioru. Różnice między różnymi wartościami były niewielkie.

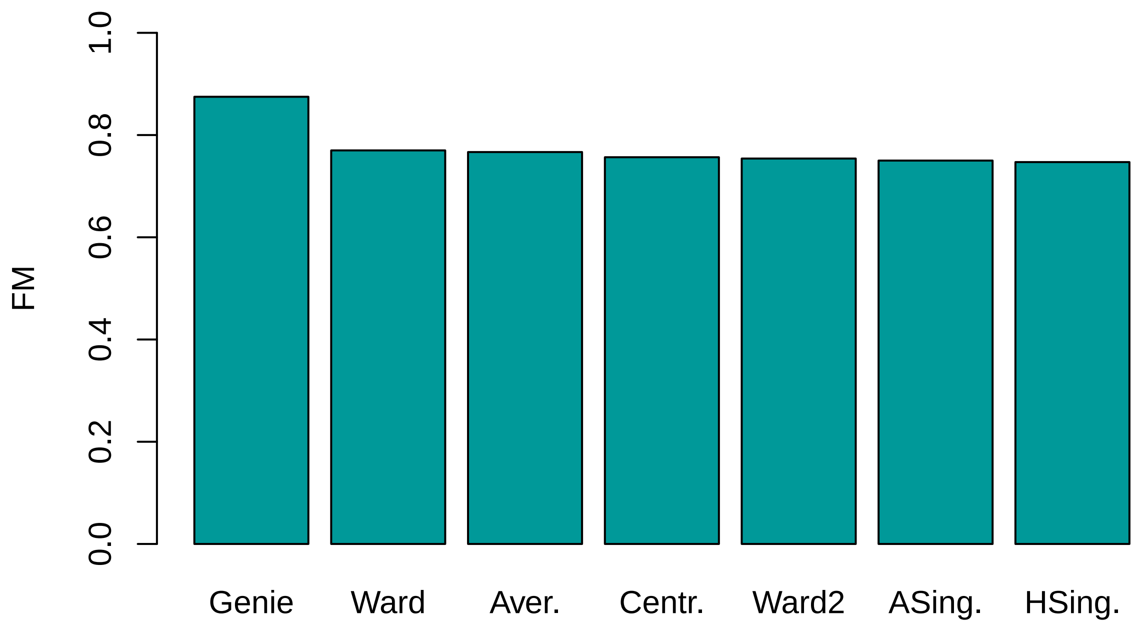
Metody funkcji `hclust()`



Spośród rodziny algorytmów dunkfcji `hclust()` najwyższą skuteczność osiągnęła metoda Warda.

Porównanie funkcji pochodzących z różnych źródeł

Czas rozstrzygnąć, która z wszystkich, 23 modyfikacji funkcji, okazała się być najlepszą. Do prezentacji wykorzystam najlepsze wersje funkcji z biblioteki `genie` oraz spośród własnego algorytmu spektralnego. Wśród wypisanych w dotychczasowych podpunktach nie znalazła się jeszcze metoda *Single* funkcji `agnes()`.



Pojedynek skuteczności funkcji do analizy spektralnej wygrywa `hclust2()` z biblioteki `genie`, osiągając średnią wartość współczynnika FM **0.87**. Jak wspomniałem, wartość domyślna współczynnika *threshold* okazała się optymalna. Na następnych miejscach zostały sklasyfikowane algorytmy z `hclust()`, jednak róznica między 1. a 2. miejscem jest znacząca i wynosi 13%.

5. Uwagi

5.1 Czas działania

```
## Unit: milliseconds
##          expr     min      lq     mean   median       uq      max neval
##   HComplete(Z)  1.2679  1.7523   20.442   17.613   35.384   75.242    60
##      HWard(Z)  1.3097  1.8976   30.261   18.194   41.758  184.580    60
##     HWard2(Z)  1.2919  1.9016   22.737   18.864   39.157   75.452    60
##    HSingle(Z)  1.1624  1.9749   18.775   28.239   32.886   39.482    60
##   HAverage(Z)  1.2420  2.1470   23.984   21.229   39.605  123.610    60
##  HMcQuitty(Z)  1.1696  1.7654   24.775   22.553   37.638  117.390    60
##    HMedian(Z)  1.1558  1.7612   23.918   20.447   38.418  116.260    60
##  HCentroid(Z)  1.3354  2.1592   26.579   21.921   45.926  104.230    60
##      GENIE(Z)  7.4962 10.6920   35.669   45.962   51.119  129.360    60
##    ASingle(Z) 10.2520 29.3670 1504.500 1177.200 2718.900 5453.300    60
##        OWN(X) 74.9050 96.9050 2232.500 2418.600 4313.500 6128.900    60
```

Czas działania testowanych funkcji był różny. Jak widać w tabeli stworzonej na podstawie analizy zbiorów z *graves* - najszybciej swoje działania wykonywały `hclust()`, które osiem swoich metod obliczały szybciej niż jedna `agnes()`. Niewiele wolniej zadziałał `genie`. Niestety, na drugim biegunie znalazł się algorytm spektralny własnej implementacji. Brak użycia pętli nie wystarczył, a najwolniej w nim działała zdecydowanie funkcja `eigen()`, poszukująca wektory oraz wartości własne laplasjanu. Krytycznie źle algorytmy zachowywały się obsługując większe zbiory danych - zawierające ponad 5000 punktów. Mnogość funkcji, szeroka gama zbiorów testowych, modyfikacje, a także i błędy, spoodowały, że obliczenia przygotowane z myślą o poniższym raporcie trwały dobre kilkadziesiąt godzin.

5.2 Algorytm *Single* z `agnes()`

Metoda *Single* pojawiła się w naszych badaniach dwukrotnie - w przypadku funkcji `hclust()` oraz `agnes()`. Nie powinny zatem dziwić orzymane podobne wyniki. Mimo bliskiej siebie skuteczności, funkcja z biblioteki `stats` działała znaczniej szybciej od `agnes()`, co niestety dyskredytuje tę drugą.

5.3 Wczytywanie plików

Jedną z zalecanych metod wczytywania plików jest wykorzystanie pętli `for`. Uznałem jednak, że w przypadku tego ćwiczenia, zbiorów benchmarkowych nie jest jeszcze aż tak wiele i wygodniej mi będzie pozostać przy dobrze mi znanej "ręcznej" metodzie wczytywania danych. Tworzona przeze mnie tablica zawierała 4 wymiary, w zależności od indeksu, skalowania, zbioru danych oraz funkcji. Jej zapis do pliku *.csv* spłaszczył ją do dwóch wymiarów.

5.4 Wykorzystane biblioteki

Oprócz wspomniany już bibliotek zawierających zaimplementowane algorytmy analizy skupień, w wykonaniu poniższego projektu pomogły mi także inne biblioteki. Są to: `plotly`, `movMF`, `igraph`, `abind`, a także - w drobnej mierze - `dplyr`.

6. Ilustracje

Poniżej prezentuję kilka najciekawszych moim zdaniem grafiki przedstawiających pracę różnych algorytmów do analizy skupień zbiorów danych.

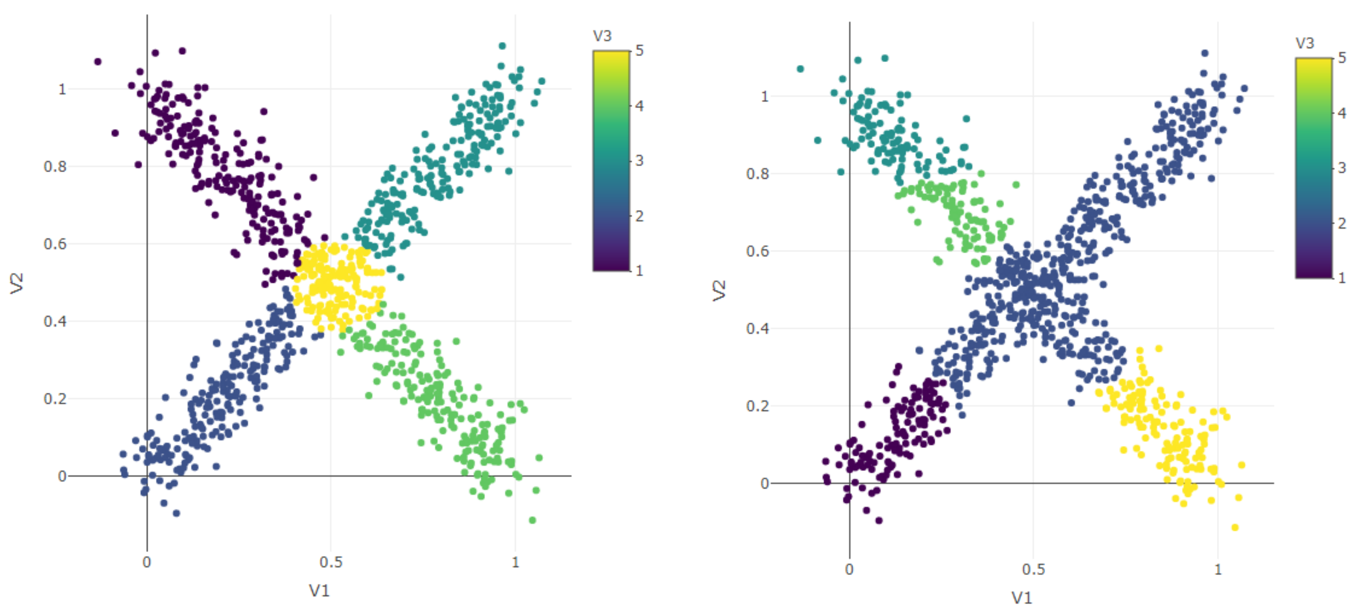


Figure 3: Algorytm *Genie* najgorzej poradził sobie ze zbiorem *fuzzy* - po lewej oczekiwany wynik, po prawej podział wg *Genie*

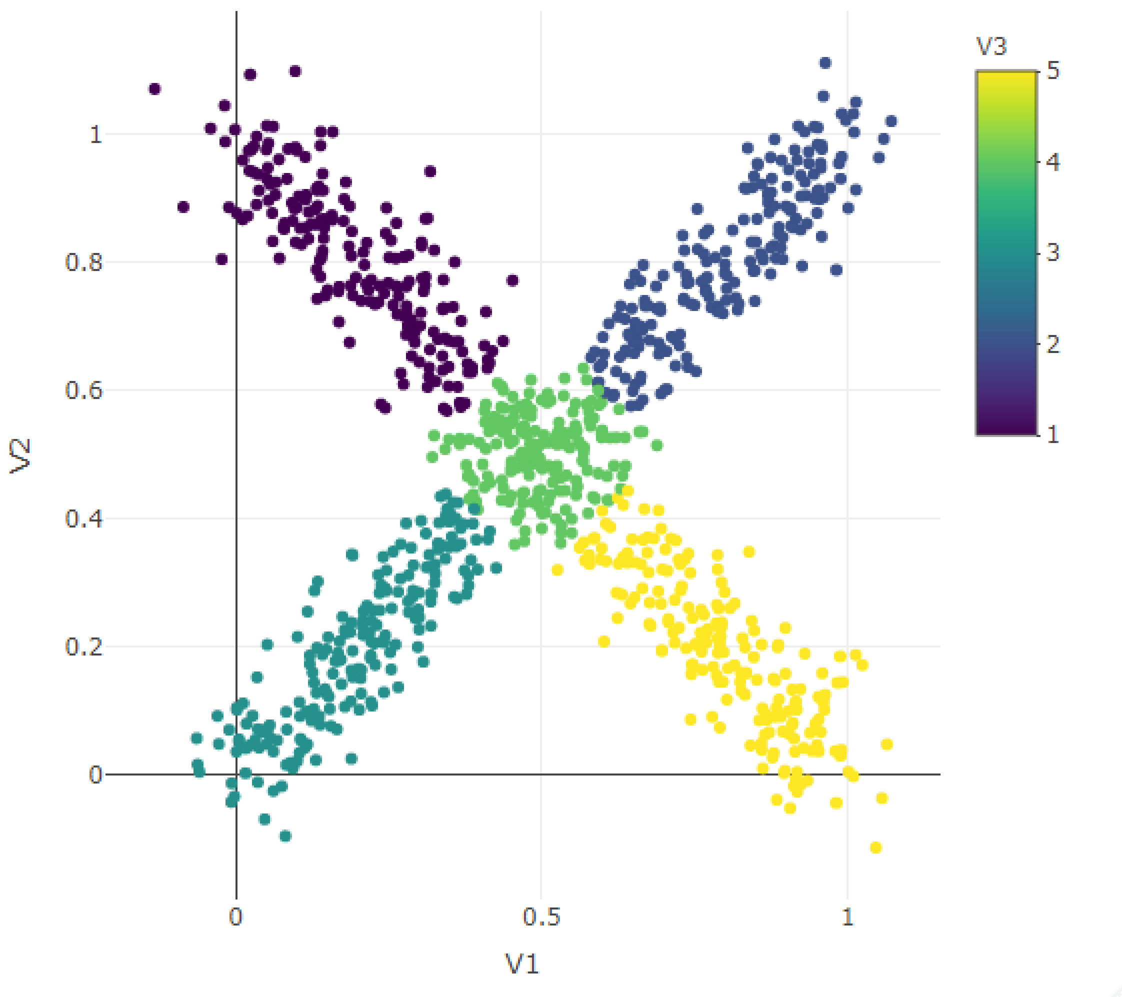


Figure 4: Najlepszą funkcją do podziału *fuzzy* okazał się własny algorytm spektralny. Otot jego wynik.

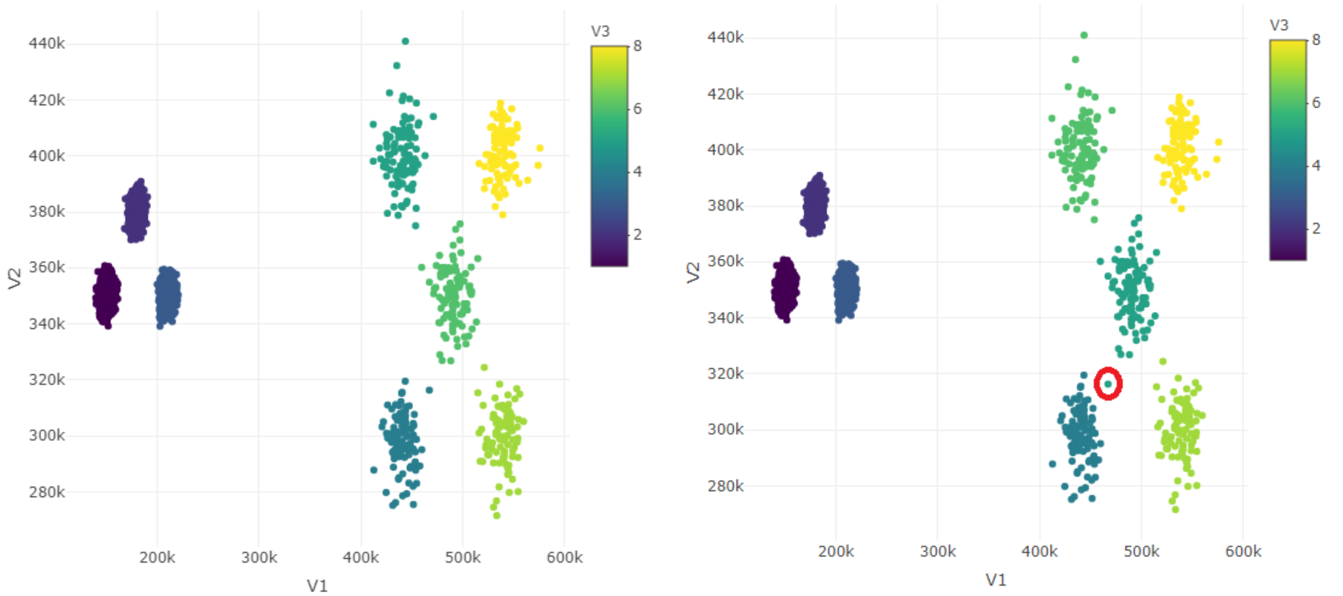


Figure 5: Oto najwyższy wynik spośród różnych od 1: metoda Warda na zbiorze *unbalance*. Różnica podziałów zaznaczona na czerwono

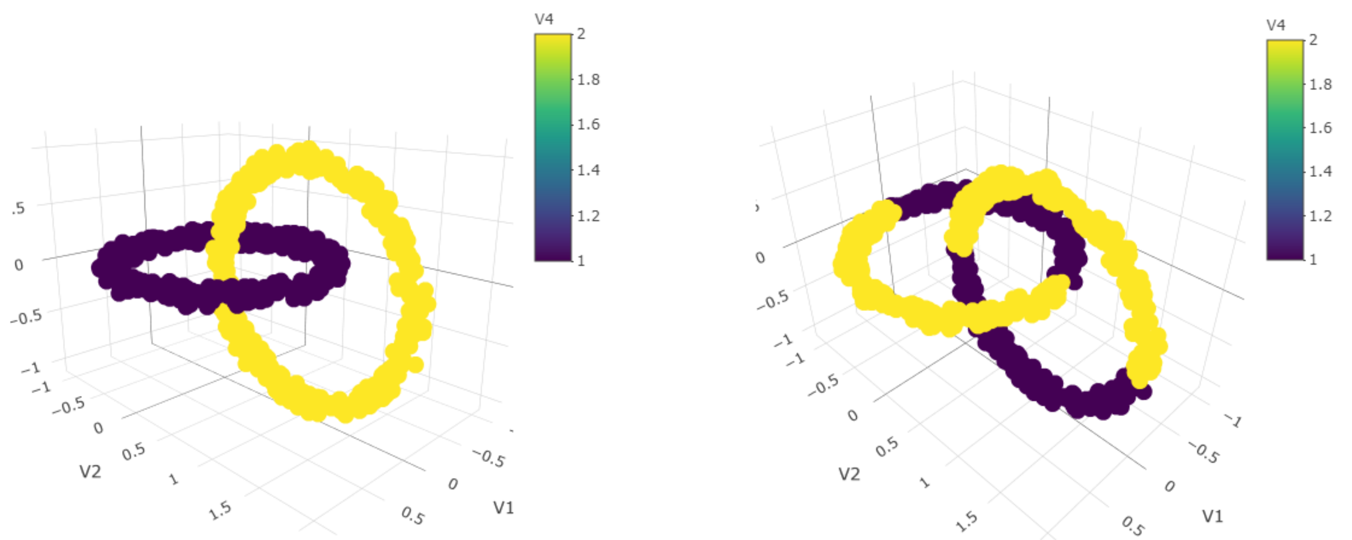


Figure 6: Najładniejszy moim zdaniem zbiór testowy - *chainlink* oraz efekt działania alg. spektralnego dla M=12 - wypadł najgorzej ze wszystkich

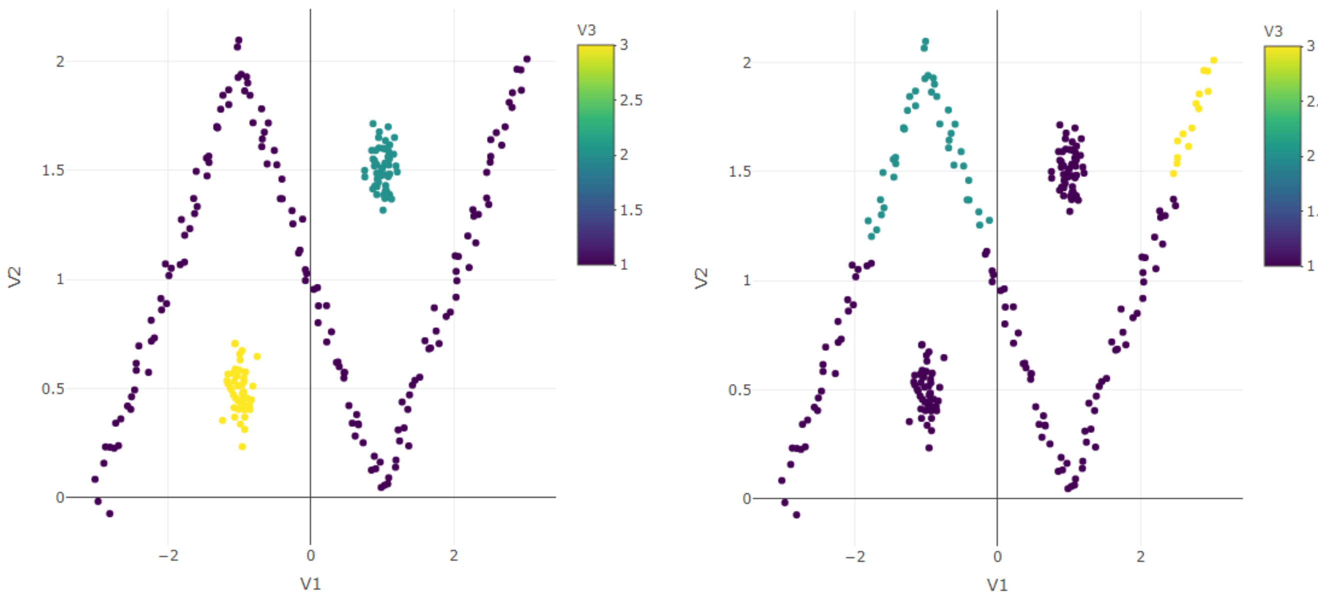


Figure 7: Oto najgorsza próba podziału spośród wszystkich, indeks Randa wyniósł jedynie -0.1. Metoda *median* na skalowanym zbiorze *zigzag*