Sprawozdanie z listy 4

Eksploracja danych

Daria Grzelak, 277533

2025-06-26

Spis treści

1 Opis danych

W niniejszym raporcie będę dokonywać dalszej analizy danych, którymi są dane `Glass` z pakietu `mlbench` (te same, co w poprzednim). Wczytam te dane poniżej, razem z potrzebnymi pakietami.

```
# Pakiety
library(mlbench) # dane Glass
library(ipred) # potrzebne funkcje
library(randomForest) # las losowy
library(e1071)
library(MASS)
library(cluster)
library(factoextra)
```

```r
# Wczytanie danych
data(Glass)

# Ustawienie ziarna generatora dla powtarzalności wyników
set.seed(123)
```

Dla przypomnienia, dane te mają 214 obserwacji oraz 10 zmiennych.

Nazwa	Typ	Opis
RI	Liczbowa	Współczynnik załamania
Na	Liczbowa	Zawartość procentowa sodu
Mg	Liczbowa	Zawartość procentowa magnezu
Al	Liczbowa	Zawartość procentowa glinu
Si	Liczbowa	Zawartość procentowa krzemu
K	Liczbowa	Zawartość procentowa potasu
Ca	Liczbowa	Zawartość procentowa wapnia
Ba	Liczbowa	Zawartość procentowa baru
Fe	Liczbowa	Zawartość procentowa żelaza
Type	Jakościowa	Typ szkła

Tabela 1: Opis cech zawartych w danych Glass

Tabela 1 zawiera opis zmiennych.

2 Zaawansowane metody klasyfikacji

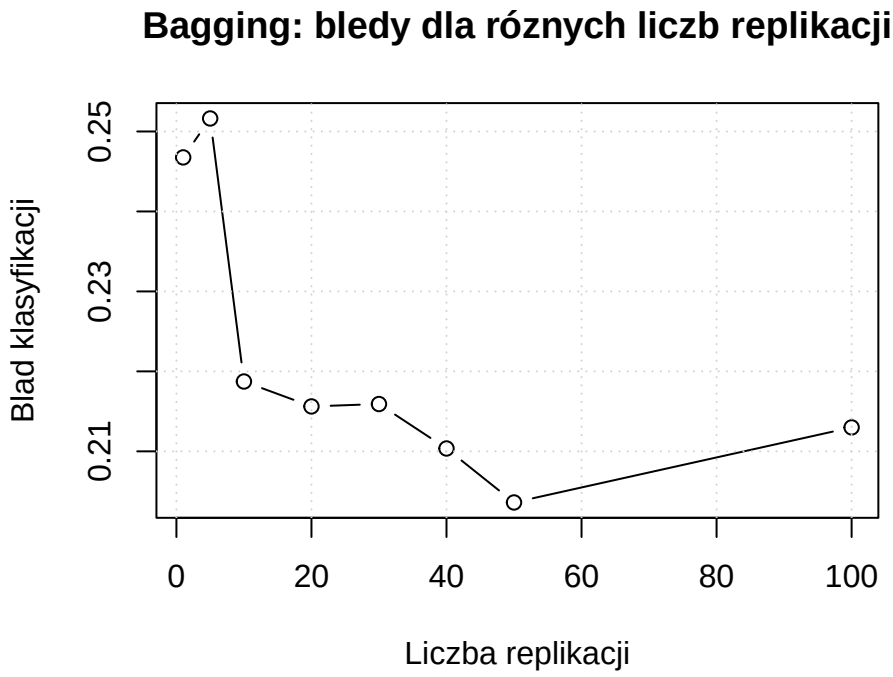
W pierwszej części niniejszego sprawozdania przedstawię wybrane zaawansowane metody klasyfikacji.

2.1 Ensemble learning

Na początku zastosuję metody operujące na rodzinach klasyfikatorów. Porównam algorytmy bagging i random forest.

2.1.1 Bagging

Pierwszy z testowanych algorytmów to algorytm bagging, polegający na wykonaniu określonej liczby bootstrapowych replikacji zbioru uczącego, skonstruowaniu dla każdej z nich klasyfikatora i wyznaczenia klasyfikatora ostatecznego, stosując zasadę głosowania większości.

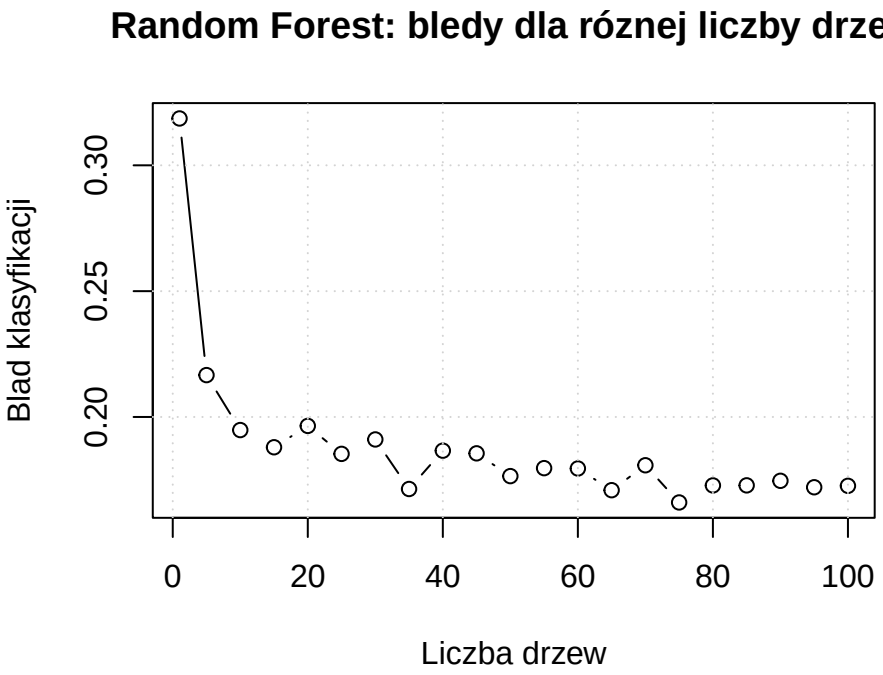


Rysunek 1: Wpływ liczby replikacji na dokładność klasyfikacji bagging

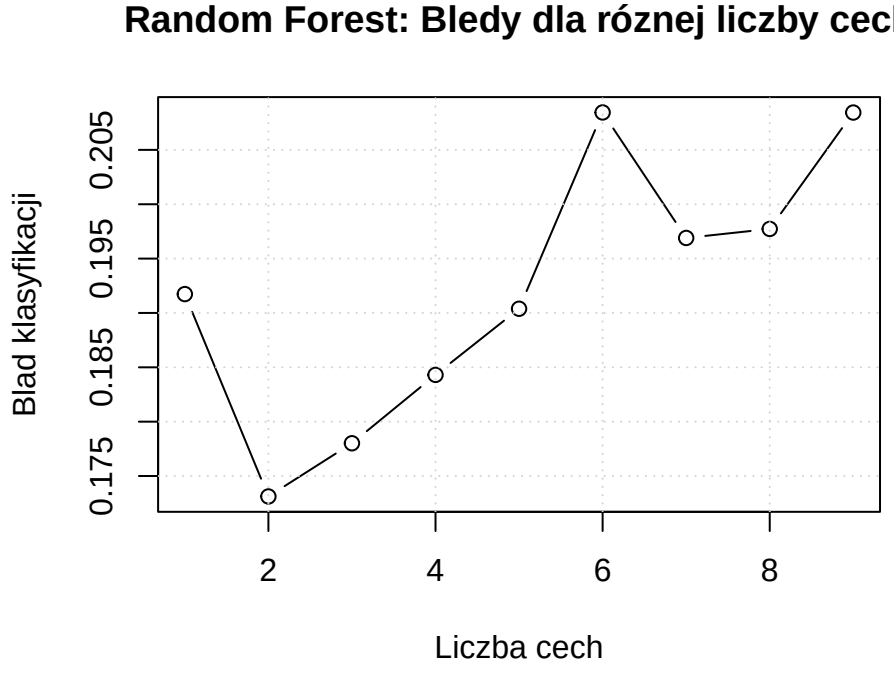
Najmniejszy błąd klasyfikacji wynosi 0.2036072 dla liczby replikacji równej 50.

2.1.2 Random Forest

Drugi z algorytmów, który przetestuję, to random forest – las losowy. Polega on na budowaniu nieskorelowanych drzew w oparciu o losowe podzbiory cech, a później wynik jest uśredniany. Porównam wyniki dla wybranych liczb drzew oraz liczb cech.



Rysunek 2: Porównanie dokładności dla liczby drzew



Rysunek 3: Porównanie dokładności dla liczby cech

Najmniejsze błędy otrzymuję kolejno dla 75 drzew (0.1660317) i dla 2 parametrów (0.1731187). Mniejszy z tych błędów wynosi 0.1660317.

2.1.3 Różnice dla enseble learning

Wykres 1 pokazuje, że liczba replikacji ma wpływ na poziom błędu, lecz nie daje ogromnej różnicy, która jest na poziomie około 0,05. Na wykresach 2 i 3 można zaobserwować, że wybór odpowiedniej liczby drzew ma dużo większe znaczenie (wahania na poziomie ponad 0,1) niż cech. Jednocześnie lasy losowe dały lepsze wyniki niż bagging.

Porównując zaawansowane schematy klasyfikacji ze zwykłym drzewem (sprawozdani z listy 3), można zauważyć znaczną poprawę jakości klasyfikacji względem zwykłych drzew (dla zwykłych drzew najmniejszy błąd, jaki udało się osiągnąć, to 0.270182, więc najmniejszy błąd dla metod ensemble learning różni się od tego o 0.1041503).

2.2 Metoda wektorów nośnych

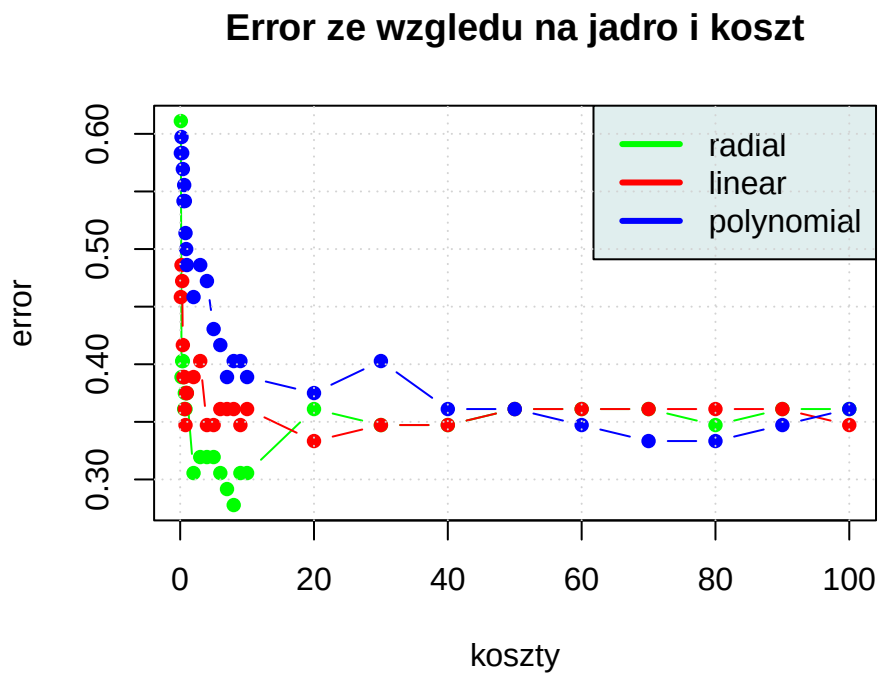
W drugiej kolejności testować będę metodę wektorów nośnych (SVM) dla jąder liniowego, radialnego i wielomianowego. Przed zastosowaniem metody przygotuję odpowiednie dane i funkcję pomocniczą.

```
# Podział na zbiory uczący i testowy
rozmiar <- nrow(Glass)
learn.ind    <- sample(1:rozmiar, 2/3*rozmiar)
glass.learn <- Glass[learn.ind,]
glass.test <- Glass[-learn.ind,]
real.labels <- glass.test$Type
rozmiar.test <- length(real.labels)

# funkcja na przewidywanie dla różnych jąder i wartości kosztu
svm_create_and_predict <- function(k, C) {
  model <- svm(Type~., data=glass.learn, kernel=k, cost=C)
  pred.svm <- predict(model, newdata=glass.test)
  acc.svm   <- sum(diag(table(pred.svm, real.labels)))/rozmiar.test
  error.svm <- 1 - acc.svm
  return (error.svm)
}

# wektor kosztów do testu
koszty <- c(seq(from=0.1, to=0.9, by=0.1), 1:10,
            seq(from=20, to=100, by=10))
```

2.2.1 Testy dla różnych jąder i wartości kosztu



Rysunek 4: Wykresy błędów dla metody SVM przy zastosowaniu różnych jąder i z różnymi wartościami kosztu

Na wykresie 4 łatwo można zauważyć, że najlepszy wynik ostatecznie został osiągnięty dla jądra radialnego (wyniósł 0.2777778 dla kosztu 8), ale jednocześnie ta metoda wynegerowała też największy ze wszystkich błędów. Jądro liniowe natomiast wykazuje najmniejsze zróżnicowanie błędów.

2.2.2 Dostraranie kosztu i gammy dla jądra radialnego

W następnej kolejności spróbuję dostroić koszt i gammę dla jądra radialnego i zobaczę, czy przyniesie to poprawę wyników.

```
# SVM { jądro radialne z domyślnymi parametrami
model.def <- svm(Type~., data=glass.learn, kernel="radial")
pred.svm.def <- predict(model.def, newdata=glass.test)
acc.svm.def <- sum(diag(table(pred.svm.def, real.labels)))/rozmiar.test
error.svm.def <- 1 - acc.svm.def

# Optymalizacja parametrów C i gamma
C.range <- 2^((-4):4)
gamma.range <- 2^((-10):4)

radial.tune <- tune(svm, train.x=glass.learn[,1:9],
                    train.y=glass.learn[,10],
```

```
                    kernel="radial",
                    ranges=list(cost=C.range, gamma=gamma.range))

# Przypisanie najlepszych parametrów
C.best <- radial.tune$best.parameters[["cost"]]
gamma.best <- radial.tune$best.parameters[["gamma"]]

# SVM { jądro radialne ze zoptymalizowanymi parametrami
model.opt <- svm(Type~., data=glass.learn, kernel="radial", cost=C.best,
                 gamma=gamma.best)
pred.svm.opt <- predict(model.opt, newdata=glass.test)
acc.svm.opt   <- sum(diag(table(pred.svm.opt, real.labels)))/rozmiar.test
error.svm.opt <- 1 - acc.svm.opt

# techniczne do automatycznego uzupełnienia, które parametry lepsze
errors.comp <- c(error.svm.def, error.svm.opt)
parametry.comp <- c("domyślne", "zoptymalizowane")
```

Ostatecznie błędy wyniosły:

- 0.375 dla parametrów domyślnych,
- 0.3333333 dla parametrów zoptymalizowanych.

Mniejszy jest błąd 0.3333333, więc lepsze okazały się parametry zoptymalizowane. Zauważyć można jednak, że błąd ten jest dość spory, podobny do zwykłych, niezaawansowanych metod klasyfikacji.

2.3 Porównanie skuteczności metod

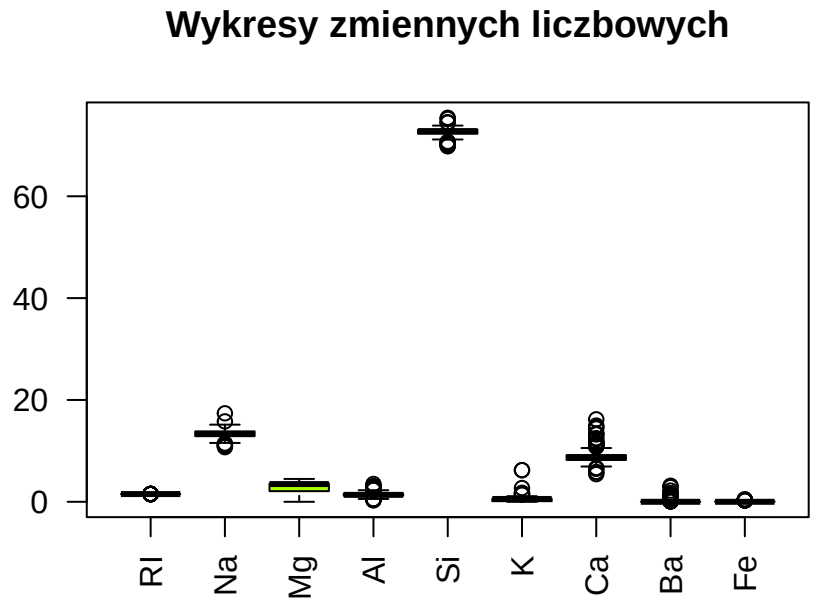
Porównując metody ensemble learning i SVM, skuteczniejsze okazały się te pierwsze, z lekką przewagą lasów losowych.

3 Analiza skupień

Drugą część niniejszego raportu stanowi analiza skupień. Będę ją wykonywać dla danych Glass, jak wcześniej.

3.1 Przygotowanie danych

Najpierw dokonam wstępnego przygotowania danych. Sprawdzę, czy dane mogą wymagać standaryzacji.



Rysunek 5: Wykresy pudełkowe dla zmiennych z danych Glass

Na wykresie 5 łatwo można zauważyć, że dane znacząco się różnią (choćby krzem ma wyższe wartości, co, oczywiście, wynika z tego, że szkło na ogół składa się przede wszystkim z niego), jednak zdecyduję się nie stosować standaryzacji. W dalszych krokach wyznaczę także macierz odmienności i przy pomocy metody PCA zredukuję wymiar, co przyda się do późniejszego rysowania wykresów.

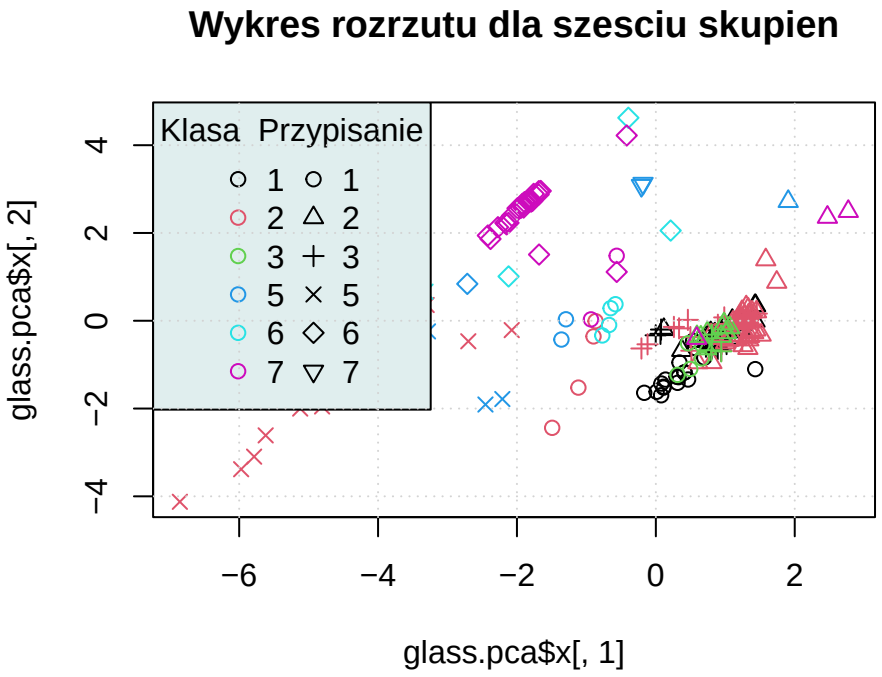
```
#dane bez etykiet
#glass.wybrane <- as.data.frame(scale(Glass[,1:9]))
glass.wybrane <- as.data.frame(Glass[,1:9])

# macierz odmienności
glass.dis.mat <- daisy(glass.wybrane)
glass.dis.mat <- as.matrix(glass.dis.mat)

# PCA
glass.pca <- prcomp(glass.wybrane, retx=TRUE)
```

3.2 Metoda grupująca (PAM)

Jako pierwszą z metod wykorzystam metodę grupującą – PAM. Najpierw przedstawię ogólny zarys dla sześciu skupień, czyli tylu, ile rzeczywiście jest klas.



Rysunek 6: Wykres rozrzutu dla sześciu skupień (PAM)

Obserwując skupiska na wykresie 6, można zauważyć brak jakiejś większej zgodności pomiędzy skupiskami i przypisaniami a rzeczywistymi klasami. Dwa największe skupiska są dość zwarte i spójne, ale jest słaba separacja – wiele klas się na siebie nakłada.

```
#Średnia wartość Silhouette
print(paste("Średnia wartość silhouette:",glass.pam$silinfo$avg.width))
```

```
## [1] "Średnia wartość silhouette: 0.247087833183455"
```

```
print("Macierz pomyłek:")
```

```
## [1] "Macierz pomyłek:"
```

```
print(table(etykietki.pam,Glass$Type))
```

```
##
## etykietki.pam  1  2  3  5  6  7
##             1 22  4  5  2  4  2
##             2 17 35  9  1  0  3
##             3 31 26  3  0  0  0
##             4  0 11  0  7  2  0
##             5  0  0  0  1  3 24
##             6  0  0  0  2  0  0
```

```
#MatchClasses
matchClasses(table(etykietki.pam,Glass$Type))
```

```
## Cases in matched pairs: 58.41 %
```

```
##   1   2   3   4   5   6
## "1" "2" "1" "2" "7" "5"
```

3.2.1 Różna liczba skupień

W następnym kroku spróbuję odnaleźć optymalną liczbę skupień.

```
# Liczby skupień od 1 do 6
clusters <- 1:6

# Funkcja pomocnicza
pam_diff_c <- function(c) {
  # PAM dla odpowiedniej liczby skupień
  glass.pam.k <- pam(x=glass.dis.mat, diss=TRUE, k=c)
  etykietki.pam.k <- glass.pam.k$clustering

  # macierz pomyłek i silhouette
  sil <- glass.pam.k$silinfo$avg.width
  tabela <- table(etykietki.pam.k,Glass$Type)

  # Wydrukuj informacje
  print(paste(c, "clusters:"))
  print(paste("Średni wskaźnik silhouette:", sil))
  print("Macierz pomyłek i wskaźnik dopasowania:")
  print(tabela)
  matchClasses(tabela)
}

# Zastosowanie funkcji
sapply(clusters, function(c) {pam_diff_c(c)})
```

```
## [1] "1 clusters:"
## [1] "Średni wskaźnik silhouette: "
## [1] "Macierz pomyłek i wskaźnik dopasowania:"
##
## etykietki.pam.k  1  2  3  5  6  7
##               1 70 76 17 13  9 29
## Cases in matched pairs: 35.51 %
## [1] "2 clusters:"
## [1] "Średni wskaźnik silhouette: 0.411215278382218"
## [1] "Macierz pomyłek i wskaźnik dopasowania:"
##
## etykietki.pam.k  1  2  3  5  6  7
##               1 70 75 17 11  5  5
```

```
##               2  0  1  0  2  4 24
## Cases in matched pairs: 46.26 %
## [1] "3 clusters:"
## [1] "Średni wskaźnik silhouette: 0.250183787375235"
## [1] "Macierz pomyłek i wskaźnik dopasowania:"
##
## etykietki.pam.k  1  2  3  5  6  7
##               1 47 43 13  9  5  5
##               2 23 33  4  2  0  0
##               3  0  0  0  2  4 24
## Cases in matched pairs: 48.6 %
## [1] "4 clusters:"
## [1] "Średni wskaźnik silhouette: 0.282787898128151"
## [1] "Macierz pomyłek i wskaźnik dopasowania:"
##
## etykietki.pam.k  1  2  3  5  6  7
##               1 17 12  2  3  1  3
##               2 41 43 12  7  4  3
##               3 12 21  3  2  0  0
##               4  0  0  0  1  4 23
## Cases in matched pairs: 48.6 %
## [1] "5 clusters:"
## [1] "Średni wskaźnik silhouette: 0.301938440085551"
## [1] "Macierz pomyłek i wskaźnik dopasowania:"
##
## etykietki.pam.k  1  2  3  5  6  7
##               1 17  2  2  0  0  3
##               2 40 43 12  4  4  3
##               3 13 21  3  2  0  0
##               4  0 10  0  6  2  0
##               5  0  0  0  1  3 23
## Cases in matched pairs: 53.27 %
## [1] "6 clusters:"
## [1] "Średni wskaźnik silhouette: 0.327811635817781"
## [1] "Macierz pomyłek i wskaźnik dopasowania:"
##
## etykietki.pam.k  1  2  3  5  6  7
##               1 17  2  2  0  0  3
##               2 40 43 12  2  4  3
##               3 13 21  3  2  0  0
##               4  0 10  0  6  2  0
##               5  0  0  0  1  3 23
##               6  0  0  0  2  0  0
## Cases in matched pairs: 54.21 %
```

```
## [[1]]
##   1
## "2"
##
## [[2]]
##   1   2
## "2" "7"
##
## [[3]]
##   1   2   3
## "1" "2" "7"
##
## [[4]]
##   1   2   3   4
## "1" "2" "2" "7"
##
## [[5]]
##   1   2   3   4   5
## "1" "2" "2" "2" "7"
##
## [[6]]
##   1   2   3   4   5   6
## "1" "2" "2" "2" "7" "5"
```

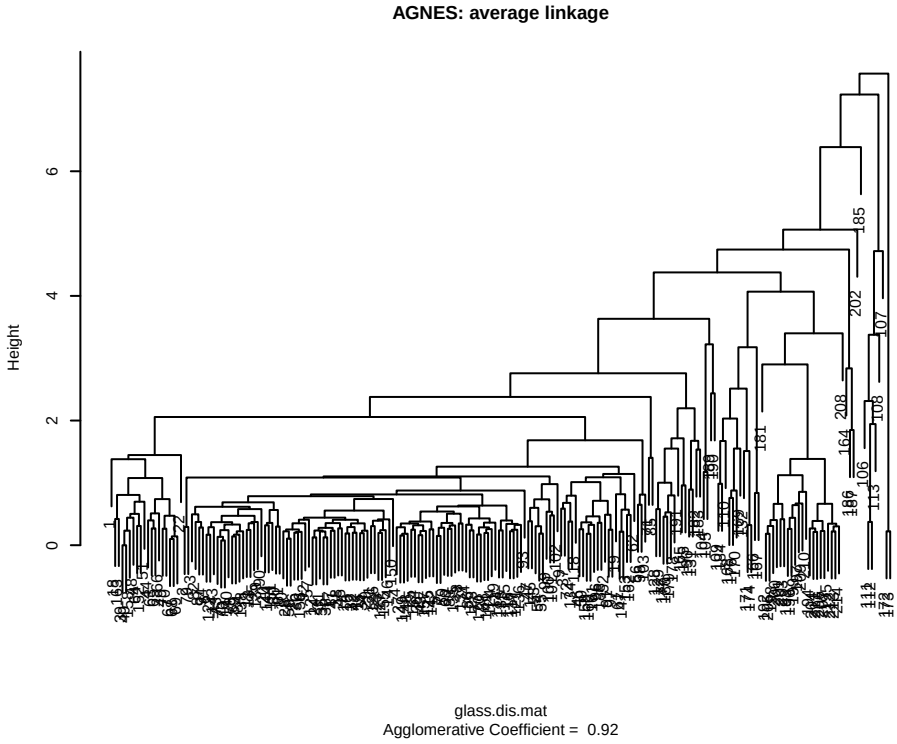
Wyniki wskazują, że wraz z liczbą skupień wzrasta dokładność według funkcji matchClasses. Wskaźnik silhouette jest za to najwyższy dla 2 skupień, choć w żadnym przypadku nie przekracza nawet wartości 0,5, więc według niego obiekty nie są zbyt dobrze przypisywane. Uwaga: dla 1 skupienia wskaźnik silhouette nie jest wyznaczany.

Będę kierować się wskaźnikiem zewnętrznym i uznam za optymalną liczbę skupień 6 skupień.

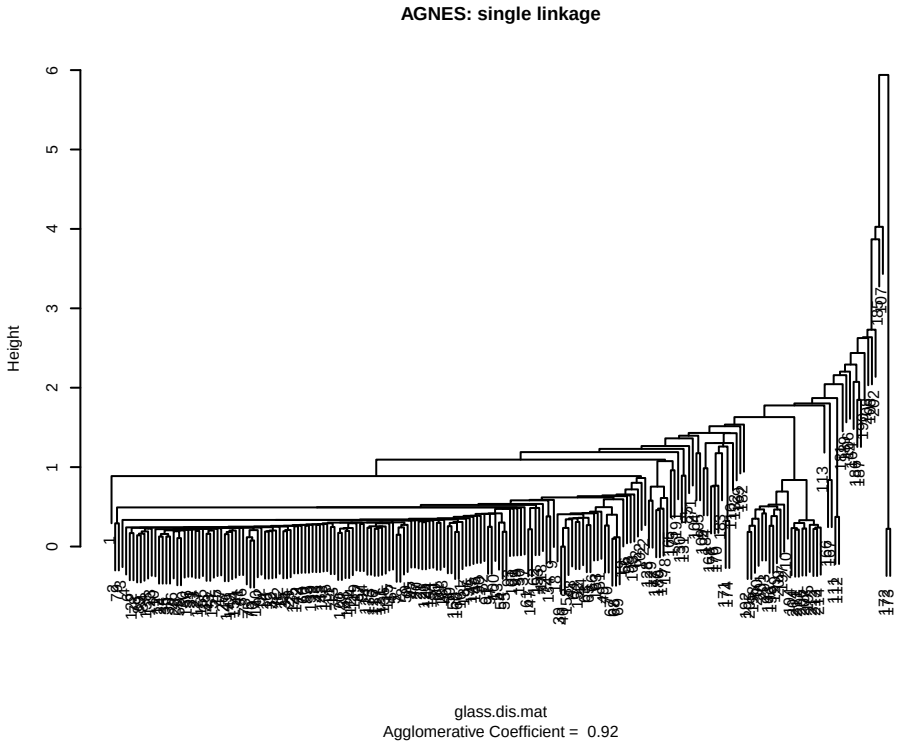
3.3 Metoda hierarchiczna (AGNES)

Drugą metodą analizy skupień, którą przeanalizuję, jest metoda hierarchiczna – AGNES. Przetestuję różne metody łączenia skupień i znajdę najlepszą z nich, wraz z optymalną liczbą skupień.

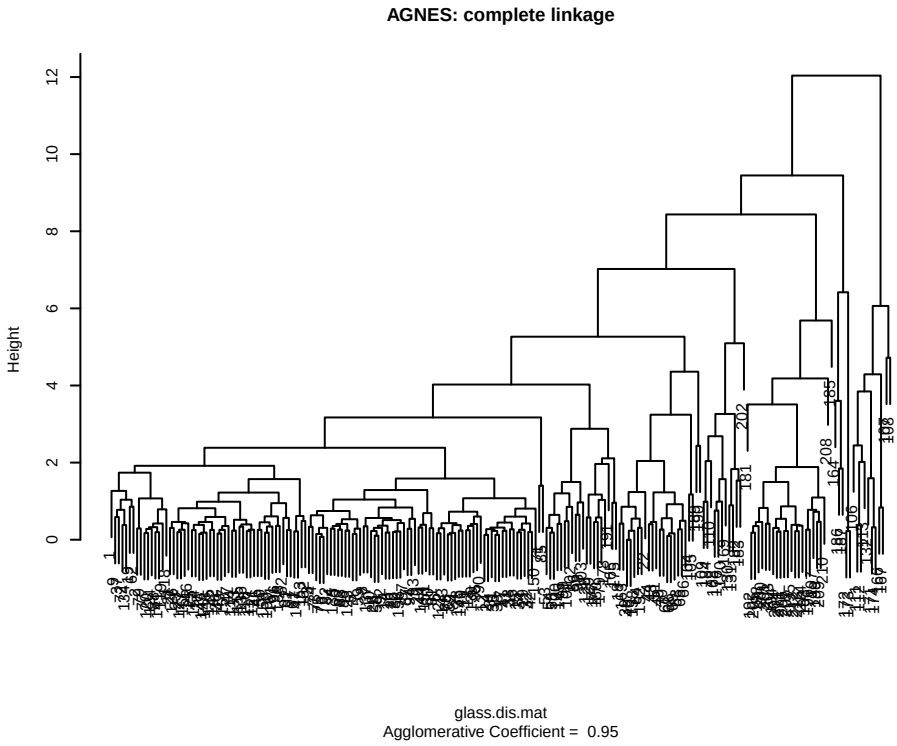
```
# Różne metody łączenia skupień
glass.agnes.avg <- agnes(x=glass.dis.mat,diss=TRUE, method="average")
glass.agnes.single <- agnes(x=glass.dis.mat,diss=TRUE, method="single")
glass.agnes.complete <- agnes(x=glass.dis.mat,diss=TRUE, method="complete")
```



Rysunek 7: Dendrogram dla metody average



Rysunek 8: Dendrogram dla metody single



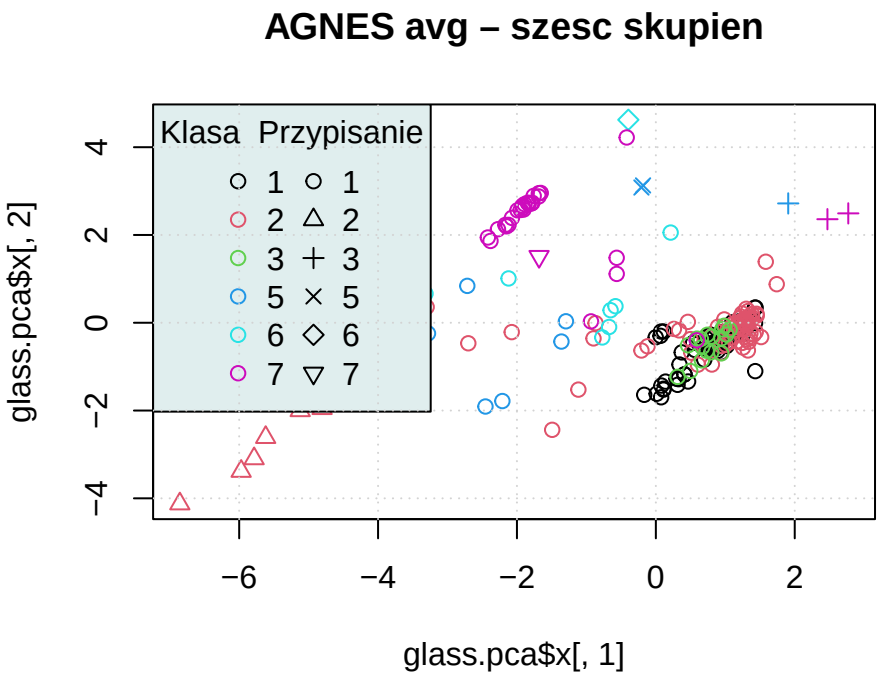
Rysunek 9: Dendrogram dla metody complete

W przypadku single linkage bardzo dużo elementów łączy się na dole drzewa, dla dwóch pozostałych metod jest więcej połączeń na różnych poziomach.

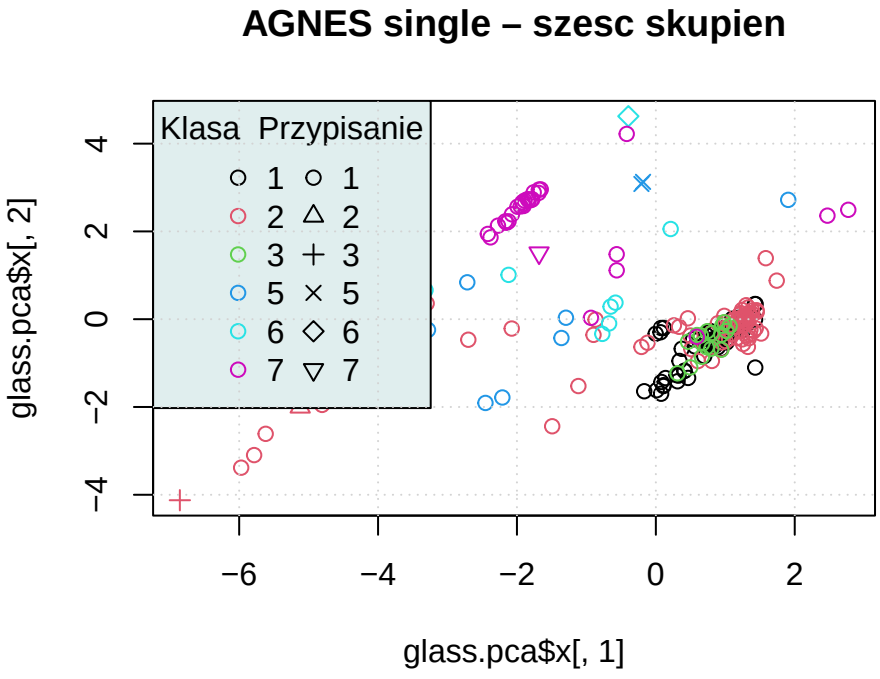
W następnych krokach będę analizować każdą metodę pod kątem optymalnej liczby skupień.

```
# Utworzenie wektorów dla każdej metody i liczby skupień
glass.avg.clusters <- sapply(clusters, function(c)
  {cutree(glass.agnes.avg, k=c)})
glass.single.clusters <- sapply(clusters, function(c)
  {cutree(glass.agnes.single, k=c)})
glass.complete.clusters <- sapply(clusters, function(c)
  {cutree(glass.agnes.complete, k=c)})
```

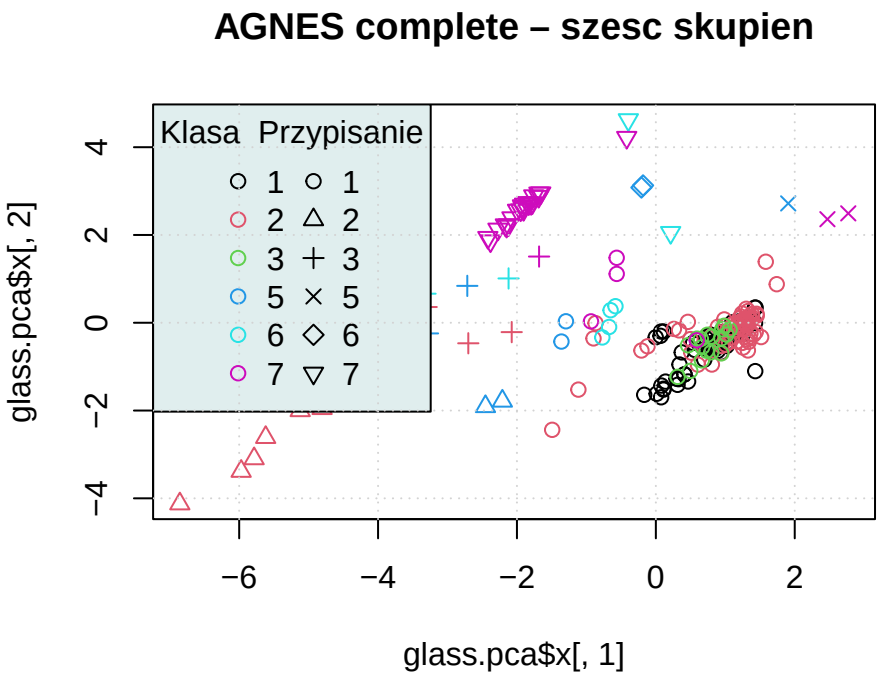
Na wykresach przedstawię wyniki dla sześciu skupień, czyli tylu, ile klas jest naprawdę.



Rysunek 10: Wykres rozrzutu dla sześciu skupień (AGNES avg)



Rysunek 11: Wykres rozrzutu dla sześciu skupień (AGNES single)



Rysunek 12: Wykres rozrzutu dla sześciu skupień (AGNES complete)

Dla metod avg i single przypisania niezbyt mają sens, gdyż w obu największych skupiskach dominuje jedno przypisanie. Inaczej jest w przypadku metody complete – wykres 12 pokazuje podział dwóch największych skupisk na dwie różne klasy. Ponadto skupisko związane z klasą 7 jest poprawnie przypisane. Za to w przypadku skupiska, gdzie zostaje przypisana klasa 1, w rzeczywistości przedstawiciele tej klasy się tam plasują, jednak pojawia się trochę przedstawicieli klas 2 i 3.

3.3.1 Optymalna liczba skupień

Tak jak wcześniej, wyznaczę wskaźniki silhouette i matchClasses, aby wybrać optymalne liczby skupień.

Avg:

```
for (n in 1:6) {
  print(paste(n,"clusters:"))
  sil.agnes.avg <- silhouette(x=glass.avg.clusters[,n], dist=glass.dis.mat)
  print(summary(sil.agnes.avg))
  tabela <- table(glass.avg.clusters[,n],Glass$Type)
  print(tabela)
  matchClasses(tabela)
}
```

```
## [1] "1 clusters:"
##    Mode    NA's
## logical       1
```

```
##
##      1  2  3  5  6  7
##   1 70 76 17 13  9 29
## Cases in matched pairs: 35.51 %
## [1] "2 clusters:"
## Silhouette of 214 units in 2 clusters from silhouette.default(x = glass.avg.clusters[
##  Cluster sizes and average silhouette widths:
##       212         2
## 0.6311138 0.9704189
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.04466 0.55588 0.70033 0.63428 0.72408 0.97053
##
##      1  2  3  5  6  7
##   1 70 76 17 11  9 29
##   2  0  0  0  2  0  0
## Cases in matched pairs: 36.45 %
## [1] "3 clusters:"
## Silhouette of 214 units in 3 clusters from silhouette.default(x = glass.avg.clusters[
##  Cluster sizes and average silhouette widths:
##       206         6         2
## 0.6124564 0.5540955 0.9701204
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.4536  0.5773  0.7202  0.6142  0.7419  0.9702
##
##      1  2  3  5  6  7
##   1 70 70 17 11  9 29
##   2  0  6  0  0  0  0
##   3  0  0  0  2  0  0
## Cases in matched pairs: 36.45 %
## [1] "4 clusters:"
## Silhouette of 214 units in 4 clusters from silhouette.default(x = glass.avg.clusters[
##  Cluster sizes and average silhouette widths:
##       205         6         2         1
## 0.5606871 0.5530293 0.9700834 0.0000000
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.4523  0.5148  0.6841  0.5617  0.7143  0.9702
##
##      1  2  3  5  6  7
##   1 70 70 17 11  8 29
##   2  0  6  0  0  0  0
##   3  0  0  0  2  0  0
##   4  0  0  0  0  1  0
```

```
## Cases in matched pairs: 36.92 %
## [1] "5 clusters:"
## Silhouette of 214 units in 5 clusters from silhouette.default(x = glass.avg.clusters[
##  Cluster sizes and average silhouette widths:
##       204         6         2         1         1
## 0.4859404 0.5406092 0.9653929 0.0000000 0.0000000
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.4519  0.4409  0.5951  0.4874  0.6320  0.9658
##
##      1  2  3  5  6  7
##   1 70 70 17 11  8 28
##   2  0  6  0  0  0  0
##   3  0  0  0  2  0  0
##   4  0  0  0  0  1  0
##   5  0  0  0  0  0  1
## Cases in matched pairs: 37.38 %
## [1] "6 clusters:"
## Silhouette of 214 units in 6 clusters from silhouette.default(x = glass.avg.clusters[
##  Cluster sizes and average silhouette widths:
##       201         6         3         2         1         1
## 0.4389977 0.5393747 0.4682106 0.9639979 0.0000000 0.0000000
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.4478  0.4267  0.5330  0.4430  0.5821  0.9641
##
##      1  2  3  5  6  7
##   1 70 70 17 10  8 26
##   2  0  6  0  0  0  0
##   3  0  0  0  1  0  2
##   4  0  0  0  2  0  0
##   5  0  0  0  0  1  0
##   6  0  0  0  0  0  1
## Cases in matched pairs: 38.32 %
```

Single:

```
for (n in 1:6) {
  print(paste(n,"clusters:"))
  sil.agnes.single <- silhouette(x=glass.single.clusters[,n],
                                 dist=glass.dis.mat)
  print(summary(sil.agnes.single))
  tabela <- table(glass.single.clusters[,n],Glass$Type)
  print(tabela)
  matchClasses(tabela)
```

```
}
```

```
## [1] "1 clusters:"
##    Mode    NA's
## logical       1
##
##      1  2  3  5  6  7
##   1 70 76 17 13  9 29
## Cases in matched pairs: 35.51 %
## [1] "2 clusters:"
## Silhouette of 214 units in 2 clusters from silhouette.default(x = glass.single.cluste
##  Cluster sizes and average silhouette widths:
##       212         2
## 0.6311138 0.9704189
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## 0.04466 0.55588 0.70033 0.63428 0.72408 0.97053
##
##      1  2  3  5  6  7
##   1 70 76 17 11  9 29
##   2  0  0  0  2  0  0
## Cases in matched pairs: 36.45 %
## [1] "3 clusters:"
## Silhouette of 214 units in 3 clusters from silhouette.default(x = glass.single.cluste
##  Cluster sizes and average silhouette widths:
##       211         1         2
## 0.5954001 0.0000000 0.9703845
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.4435  0.5131  0.7040  0.5961  0.7266  0.9705
##
##      1  2  3  5  6  7
##   1 70 75 17 11  9 29
##   2  0  1  0  0  0  0
##   3  0  0  0  2  0  0
## Cases in matched pairs: 36.45 %
## [1] "4 clusters:"
## Silhouette of 214 units in 4 clusters from silhouette.default(x = glass.single.cluste
##  Cluster sizes and average silhouette widths:
##       210         1         2         1
## 0.5367710 0.0000000 0.9703502 0.0000000
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.4424  0.4659  0.6637  0.5358  0.6935  0.9705
```

```
##
##      1  2  3  5  6  7
##   1 70 75 17 11  8 29
##   2  0  1  0  0  0  0
##   3  0  0  0  2  0  0
##   4  0  0  0  0  1  0
## Cases in matched pairs: 36.92 %
## [1] "5 clusters:"
## Silhouette of 214 units in 5 clusters from silhouette.default(x = glass.single.cluste
##  Cluster sizes and average silhouette widths:
##       209         1         2         1         1
## 0.4497149 0.0000000 0.9653929 0.0000000 0.0000000
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.4421  0.3549  0.5680  0.4482  0.6065  0.9658
##
##      1  2  3  5  6  7
##   1 70 75 17 11  8 28
##   2  0  1  0  0  0  0
##   3  0  0  0  2  0  0
##   4  0  0  0  0  1  0
##   5  0  0  0  0  0  1
## Cases in matched pairs: 37.38 %
## [1] "6 clusters:"
## Silhouette of 214 units in 6 clusters from silhouette.default(x = glass.single.cluste
##  Cluster sizes and average silhouette widths:
##       208         1         1         2         1         1
## 0.4538939 0.0000000 0.0000000 0.9653929 0.0000000 0.0000000
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.5982  0.3607  0.5746  0.4502  0.6127  0.9658
##
##      1  2  3  5  6  7
##   1 70 74 17 11  8 28
##   2  0  1  0  0  0  0
##   3  0  1  0  0  0  0
##   4  0  0  0  2  0  0
##   5  0  0  0  0  1  0
##   6  0  0  0  0  0  1
## Cases in matched pairs: 37.38 %
```

Complete:

```
for (n in 1:6) {
  print(paste(n,"clusters:"))
```

```
  sil.agnes.complete <- silhouette(x=glass.complete.clusters[,n],
                                   dist=glass.dis.mat)
  print(summary(sil.agnes.complete))
  tabela <- table(glass.complete.clusters[,n],Glass$Type)
  print(tabela)
  matchClasses(tabela)
}
```

```
## [1] "1 clusters:"
##    Mode    NA's
## logical       1
##
##      1  2  3  5  6  7
##   1 70 76 17 13  9 29
## Cases in matched pairs: 35.51 %
## [1] "2 clusters:"
## Silhouette of 214 units in 2 clusters from silhouette.default(x = glass.complete.clus
##  Cluster sizes and average silhouette widths:
##       203        11
## 0.5786264 0.4300490
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.2693  0.5121  0.6765  0.5710  0.7124  0.7338
##
##      1  2  3  5  6  7
##   1 70 69 17  9  9 29
##   2  0  7  0  4  0  0
## Cases in matched pairs: 35.98 %
## [1] "3 clusters:"
## Silhouette of 214 units in 3 clusters from silhouette.default(x = glass.complete.clus
##  Cluster sizes and average silhouette widths:
##       198        11         5
## 0.5584469 0.4219389 0.1824876
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.2562  0.4663  0.6380  0.5426  0.6812  0.7081
##
##      1  2  3  5  6  7
##   1 70 69 17  6  9 27
##   2  0  7  0  4  0  0
##   3  0  0  0  3  0  2
## Cases in matched pairs: 37.38 %
## [1] "4 clusters:"
## Silhouette of 214 units in 4 clusters from silhouette.default(x = glass.complete.clus
```

```
##  Cluster sizes and average silhouette widths:
##       174        11         5        24
## 0.5548566 0.4039026 0.1673662 0.6067162
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.3193  0.5402  0.6413  0.5439  0.6798  0.7111
##
##      1  2  3  5  6  7
##   1 70 69 17  6  7  5
##   2  0  7  0  4  0  0
##   3  0  0  0  3  0  2
##   4  0  0  0  0  2 22
## Cases in matched pairs: 47.66 %
## [1] "5 clusters:"
## Silhouette of 214 units in 5 clusters from silhouette.default(x = glass.complete.clus
##  Cluster sizes and average silhouette widths:
##       162        11        12         5        24
## 0.6399321 0.1408518 0.3061396 0.1461464 0.5158268
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.2246  0.4874  0.6678  0.5701  0.7254  0.7637
##
##      1  2  3  5  6  7
##   1 70 65 17  2  4  4
##   2  0  7  0  4  0  0
##   3  0  4  0  4  3  1
##   4  0  0  0  3  0  2
##   5  0  0  0  0  2 22
## Cases in matched pairs: 49.53 %
## [1] "6 clusters:"
## Silhouette of 214 units in 6 clusters from silhouette.default(x = glass.complete.clus
##  Cluster sizes and average silhouette widths:
##       162        11        12         3         2        24
## 0.6259146 0.1408518 0.3061396 0.4274931 0.9639979 0.5093771
## Individual silhouette widths:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.2246  0.5061  0.6472  0.5704  0.7133  0.9641
##
##      1  2  3  5  6  7
##   1 70 65 17  2  4  4
##   2  0  7  0  4  0  0
##   3  0  4  0  4  3  1
##   4  0  0  0  1  0  2
##   5  0  0  0  2  0  0
##   6  0  0  0  0  2 22
```

```
## Cases in matched pairs: 50 %
```

W związku z powyższym można zauważyć następujące rzeczy:

- najwyższy średni silhouette dla avg – 0,63428 przy 2 klastrach,
- najwyższy wskaźnik zewnętrzny dla avg – 38,32% przy 6 klastrach,
- najwyższy średni silhouette dla single – 37,38% przy 6 klastrach,
- najwyższy wskaźnik zewnętrzny dla single – 0,63428 przy 2 klastrach,
- najwyższy średni silhouette dla complete – 0,571 przy 2 klastrach,
- najwyższy wskaźnik zewnętrzny dla complete – 50% przy 6 klastrach.

Zatem najbardziej obiecujące jest użycie metody łączenia complete dla 6 klastrów, czyli tylu, ile jest klas.

3.4 Charakterystyki skupień

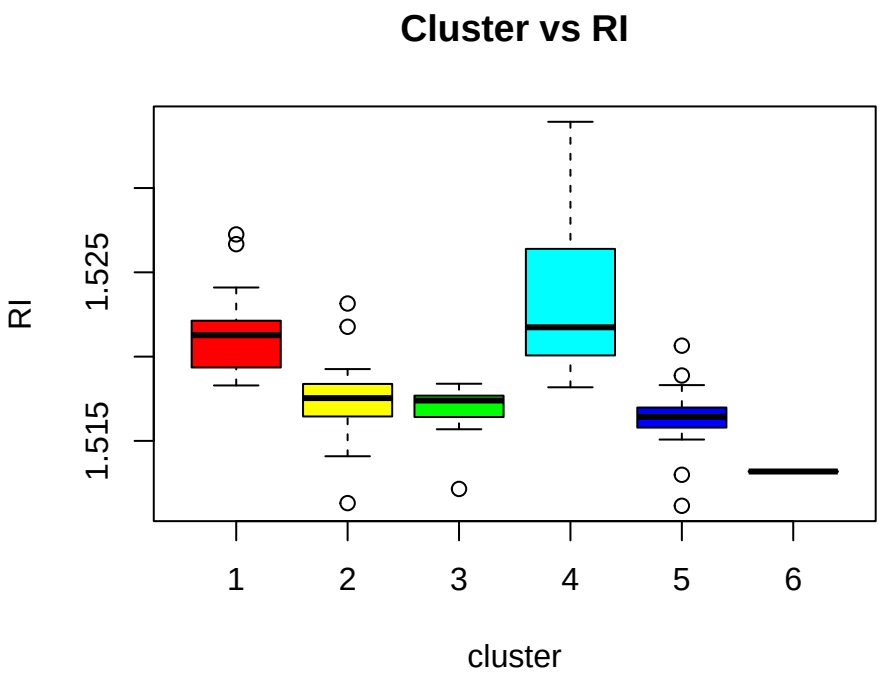
W obu przypadkach najoptymalniejsze okazało się wykorzystanie 6 skupień, tak więc dla nich będę robić. Wykresy dla tej liczby zostały przedstawione wcześniej, zatem przejdę jedynie do analizy samych skupień.

3.4.1 PAM

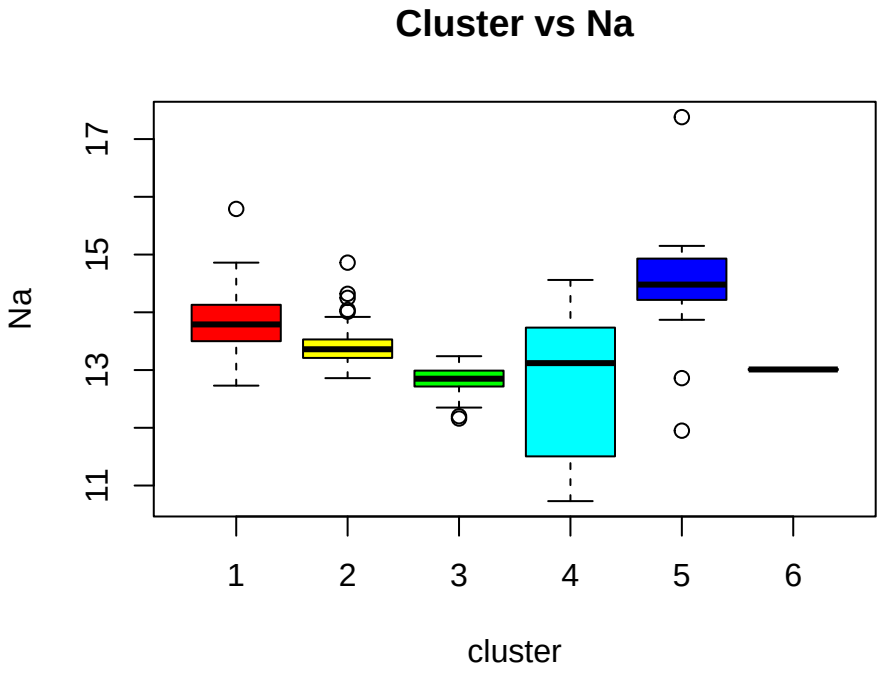
Najpierw porównam cechy dla obiektów należących do poszczególnych skupień i zrobię to na podstawie wykresów pudełkowych. Najpierw jednak dokleję do danych glass przypisane etykietki, by było łatwiej.

```
#Utworzenie odpowiedniego zbioru
glass.z.przypisaniami <- as.data.frame(cbind(glass.wybrane,
                                             glass.pam$clustering))
names(glass.z.przypisaniami)[10] <- "cluster"
```

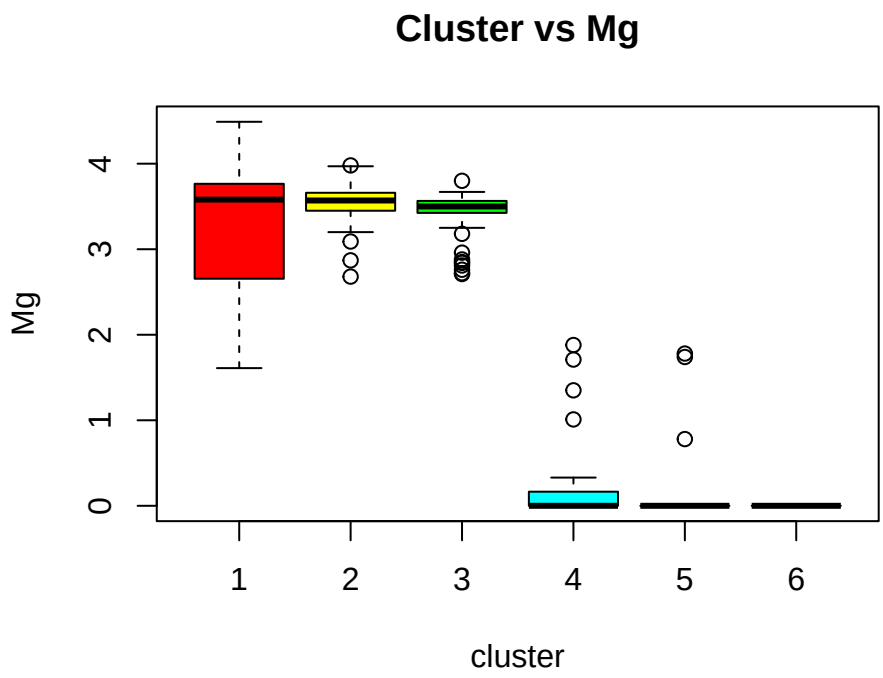
Poniżej natomiast przedstawię wykresy dla wszystkich cech.



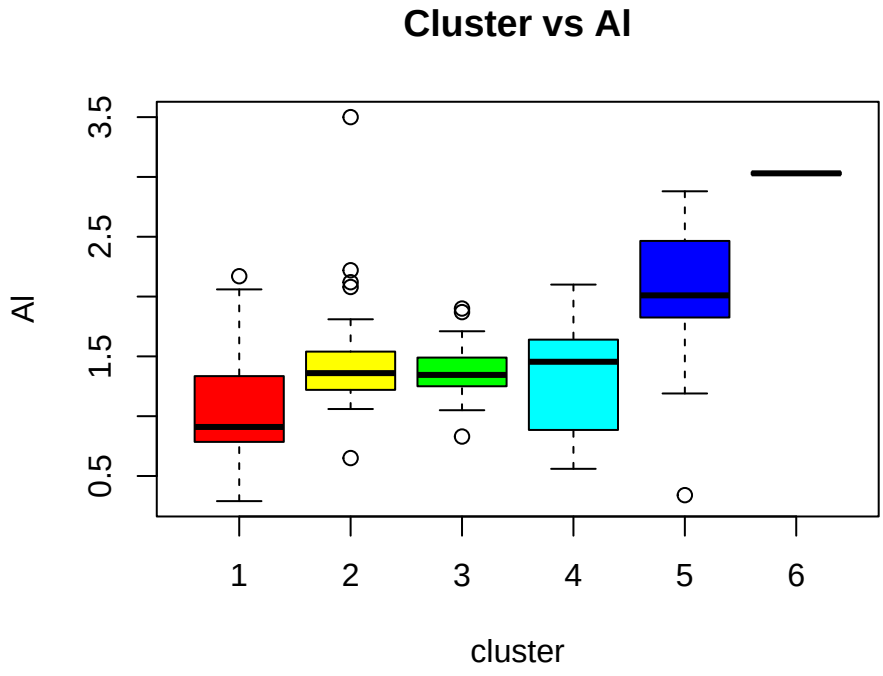
Rysunek 13: Wykresy pudełkowe dla różnych klastrów, dla cechy RI



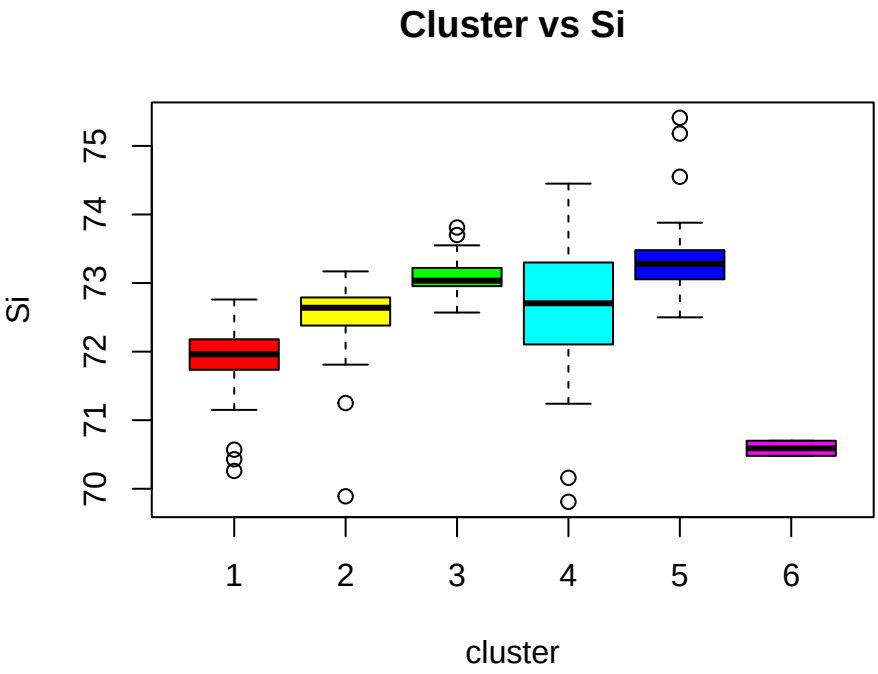
Rysunek 14: Wykresy pudełkowe dla różnych klastrów, dla cechy Na



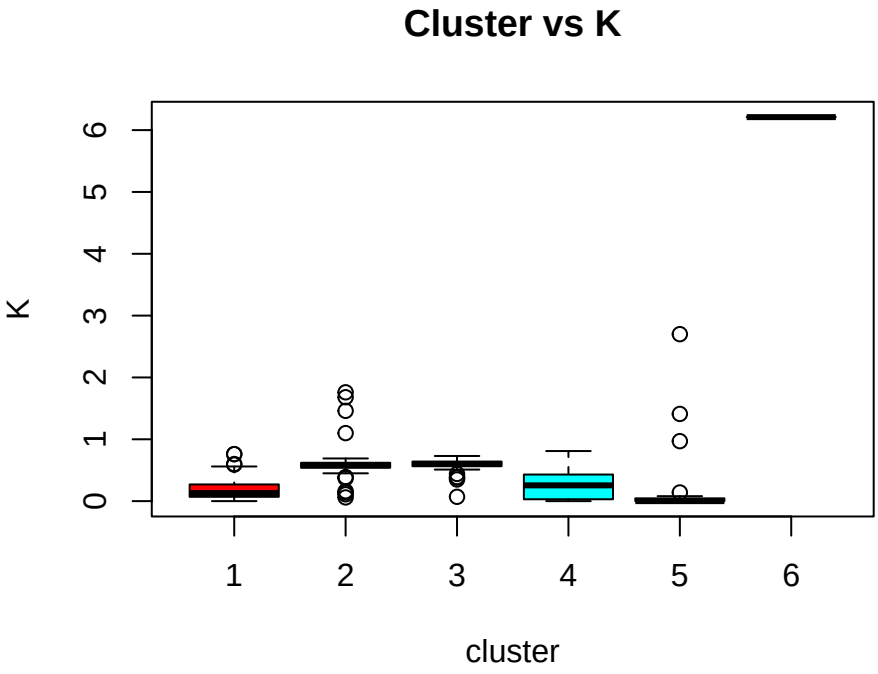
Rysunek 15: Wykresy pudełkowe dla różnych klastrów, dla cechy Mg



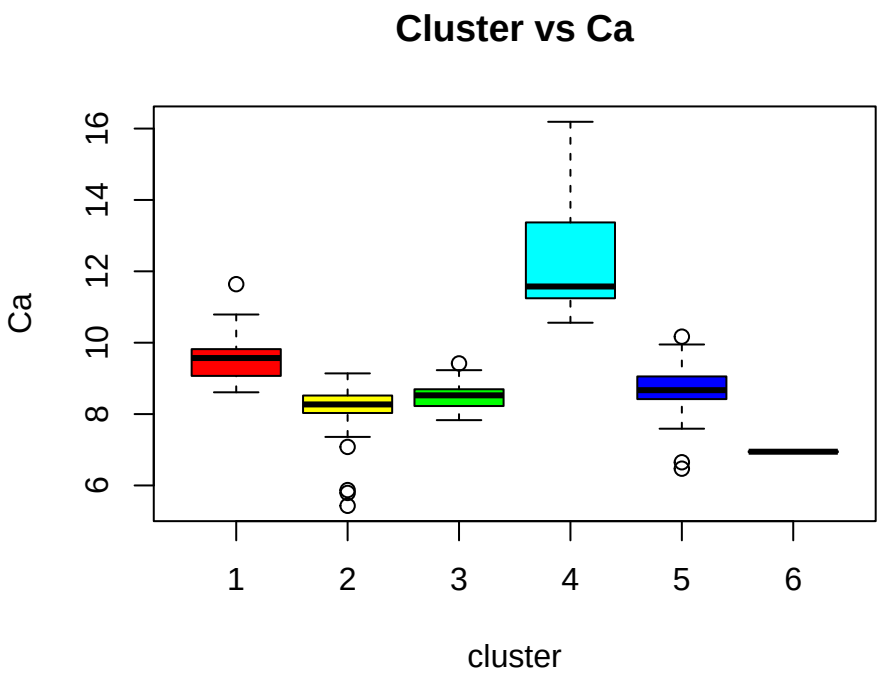
Rysunek 16: Wykresy pudełkowe dla różnych klastrów, dla cechy Al



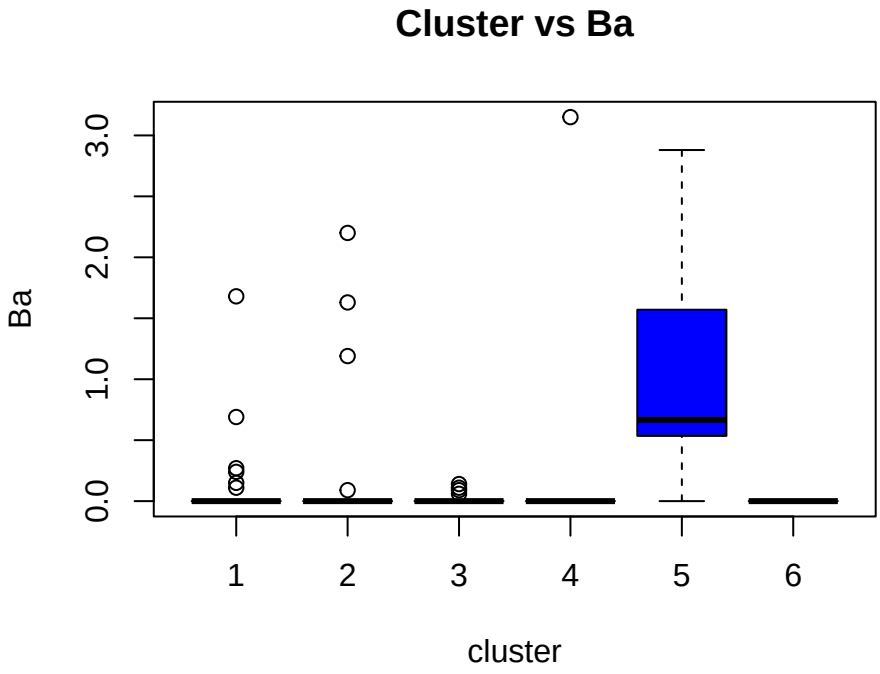
Rysunek 17: Wykresy pudełkowe dla różnych klastrów, dla cechy Si



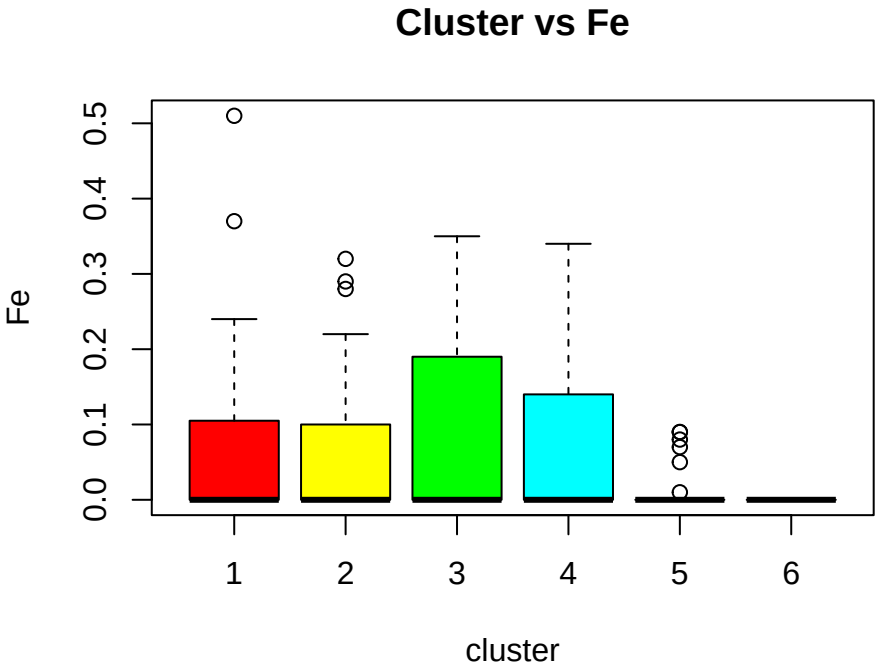
Rysunek 18: Wykresy pudełkowe dla różnych klastrów, dla cechy K



Rysunek 19: Wykresy pudełkowe dla różnych klastrów, dla cechy Ca



Rysunek 20: Wykresy pudełkowe dla różnych klastrów, dla cechy Ba



Rysunek 21: Wykresy pudełkowe dla różnych klastrów, dla cechy Fe

Cechy są bardziej zróżnicowane względem klastrów niż względem rzeczywistych klas (patrz: sprawozdanie 3), a uwagę zwracają szczególnie cechy RI, Na i Si, zwłaszcza Si jest brdziej zróżnicowana niż przy oryginalnym podziale.

Sprawdzę także, które z obiektów są medoidami.

```
# Wyświetlenie medoidów
Glass[glass.pam$medoids,]
```

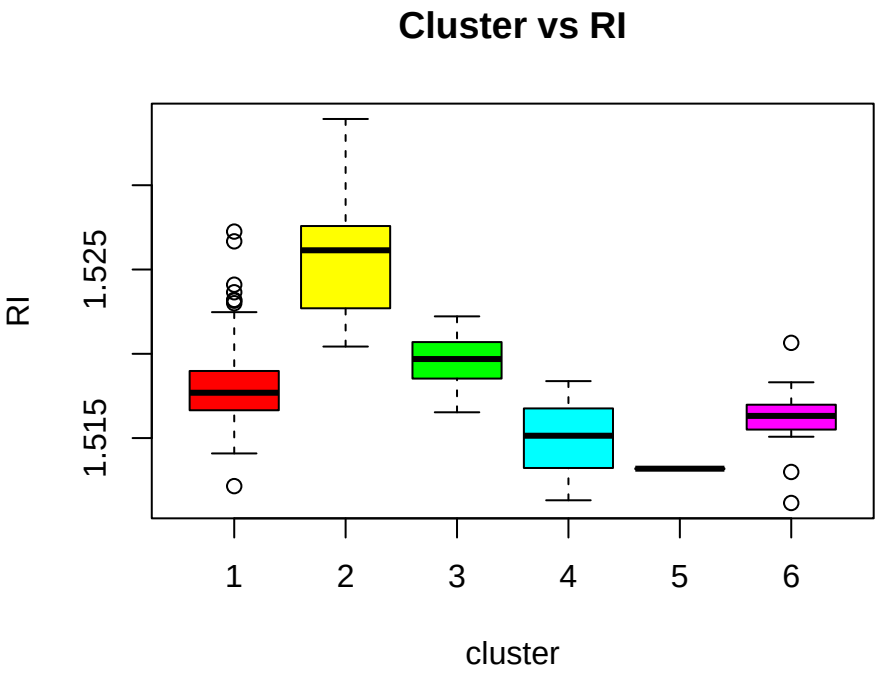
```
##           RI    Na   Mg   Al    Si    K    Ca   Ba Fe Type
## 66  1.52099 13.69 3.59 1.12 71.96 0.09  9.40 0.00  0    1
## 148 1.51610 13.33 3.53 1.34 72.67 0.56  8.33 0.00  0    3
## 28  1.51721 12.87 3.48 1.33 73.04 0.56  8.43 0.00  0    1
## 171 1.52369 13.44 0.00 1.58 72.22 0.32 12.24 0.00  0    5
## 198 1.51727 14.70 0.00 2.34 73.28 0.00  8.95 0.66  0    7
## 172 1.51316 13.02 0.00 3.04 70.48 6.21  6.96 0.00  0    5
```

W tym przypadku medoidy należą do klas 1, 3, 5 oraz 7, a więc nie wszystkie są uwzględnione w tym przypadku. Dla tych obiektów dość mocno zróżnicowana jest zawartość wapnia, a także potasu. Ponasto obiekt z klasy 7 jest jedynym, który ma niezerową zawartość baru.

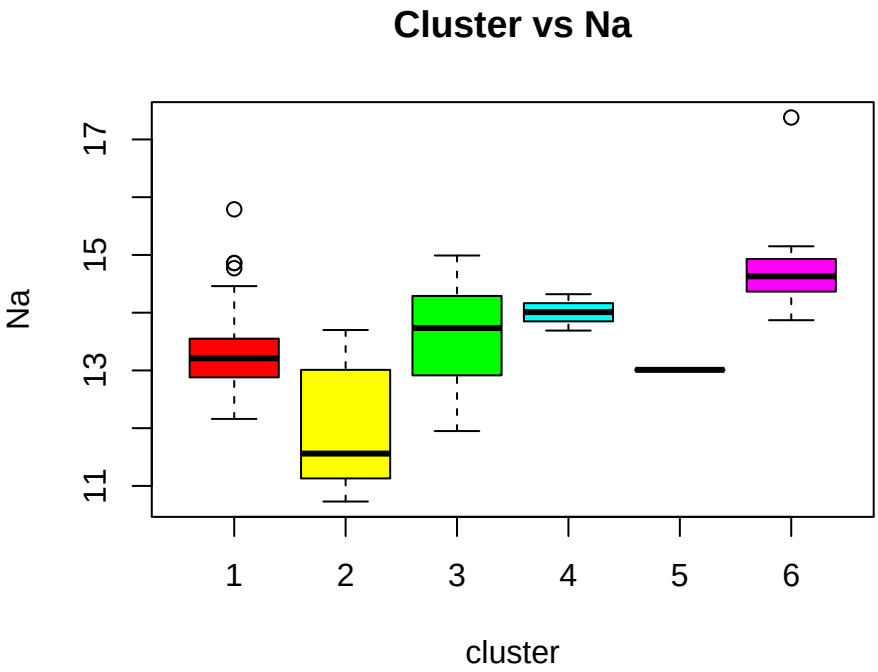
3.4.2 AGNES

Teraz taką samą analizę cech wykonam dla metody AGNES. Wybiorę metodę complete, dla 6 skupień.

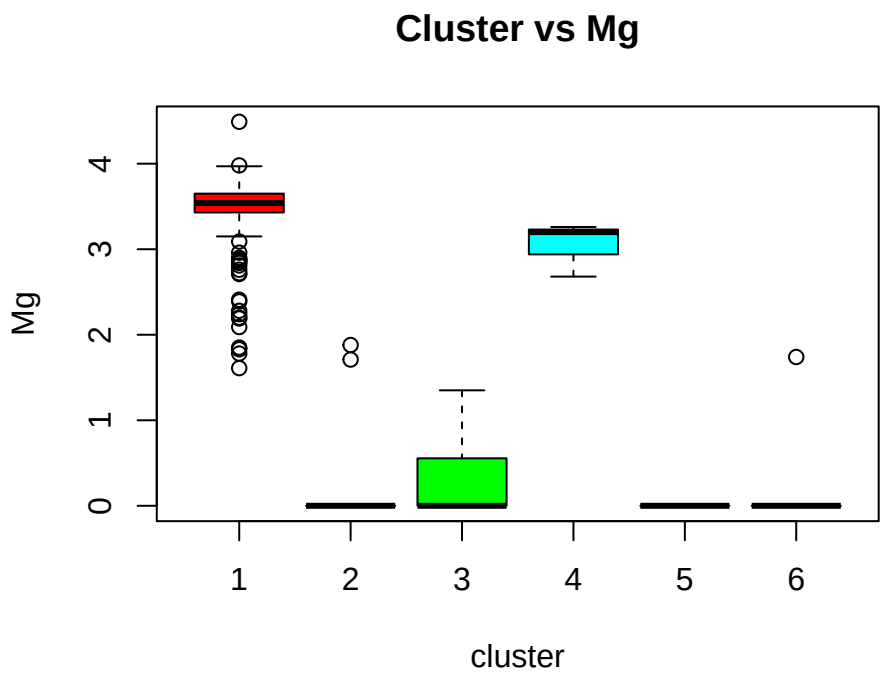
```r
#Utworzenie odpowiedniego zbioru
glass.z.przypisaniami2 <- as.data.frame(cbind(glass.wybrane,
                                              glass.complete.clusters[,6]))
names(glass.z.przypisaniami2)[10] <- "cluster"
```



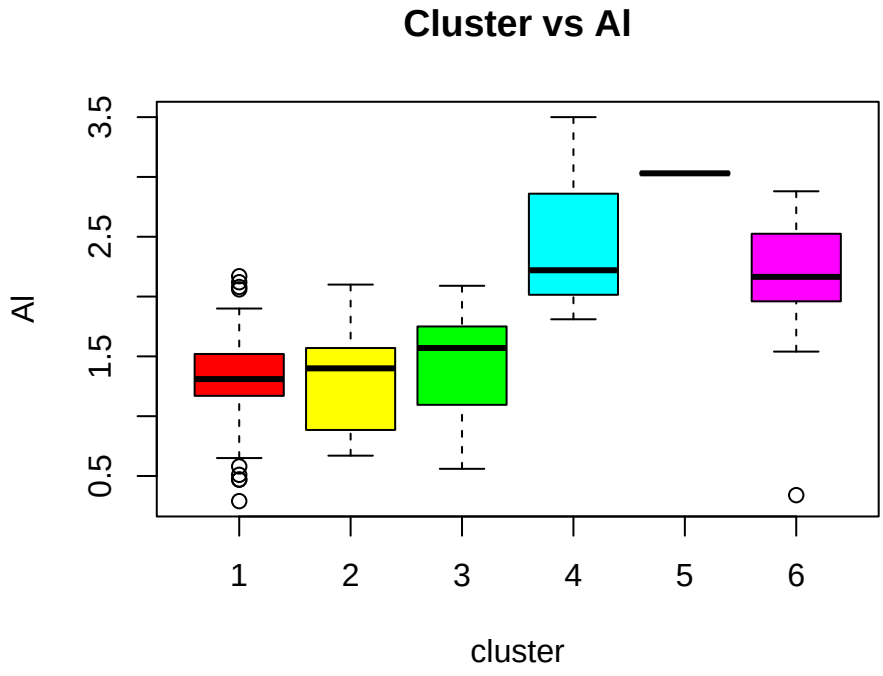
Rysunek 22: Wykresy pudełkowe dla różnych klastrów, dla cechy RI



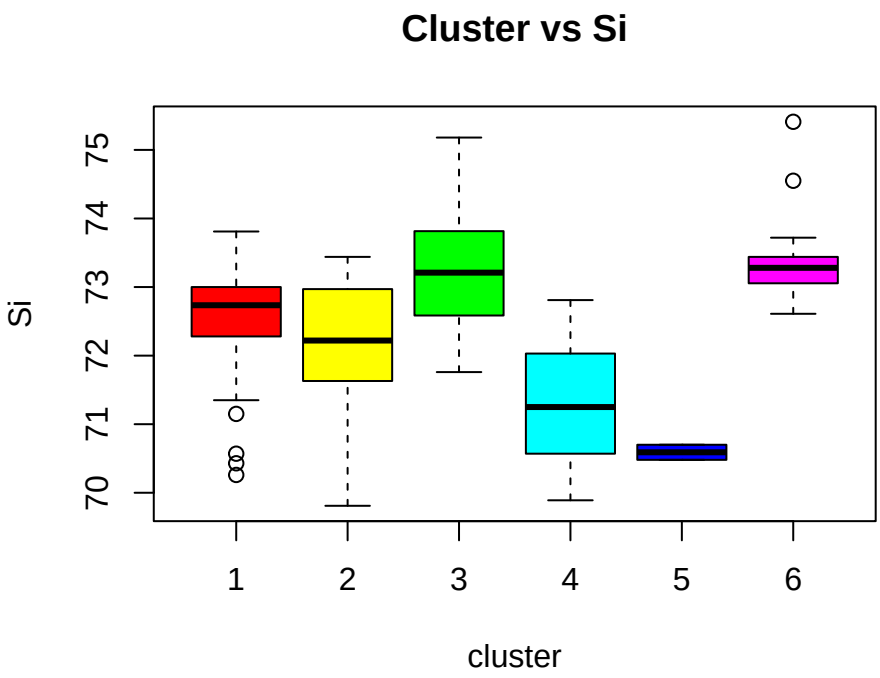
Rysunek 23: Wykresy pudełkowe dla różnych klastrów, dla cechy Na



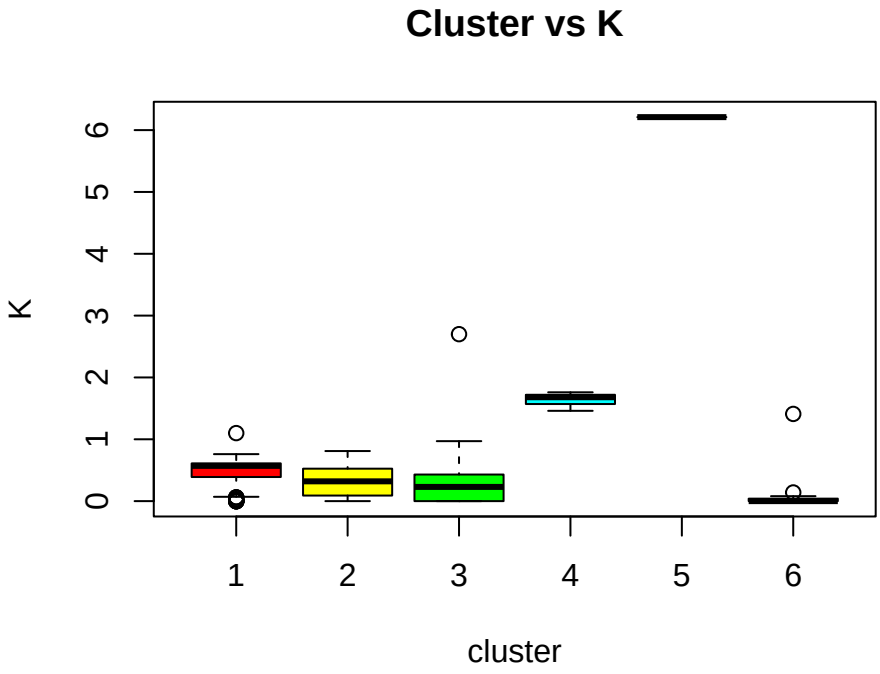
Rysunek 24: Wykresy pudełkowe dla różnych klastrów, dla cechy Mg



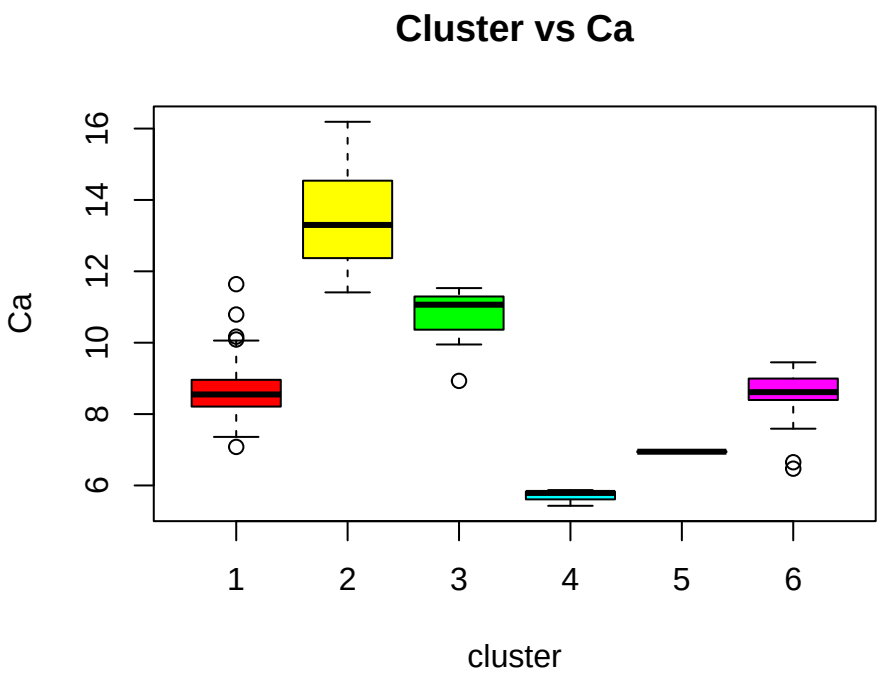
Rysunek 25: Wykresy pudełkowe dla różnych klastrów, dla cechy Al



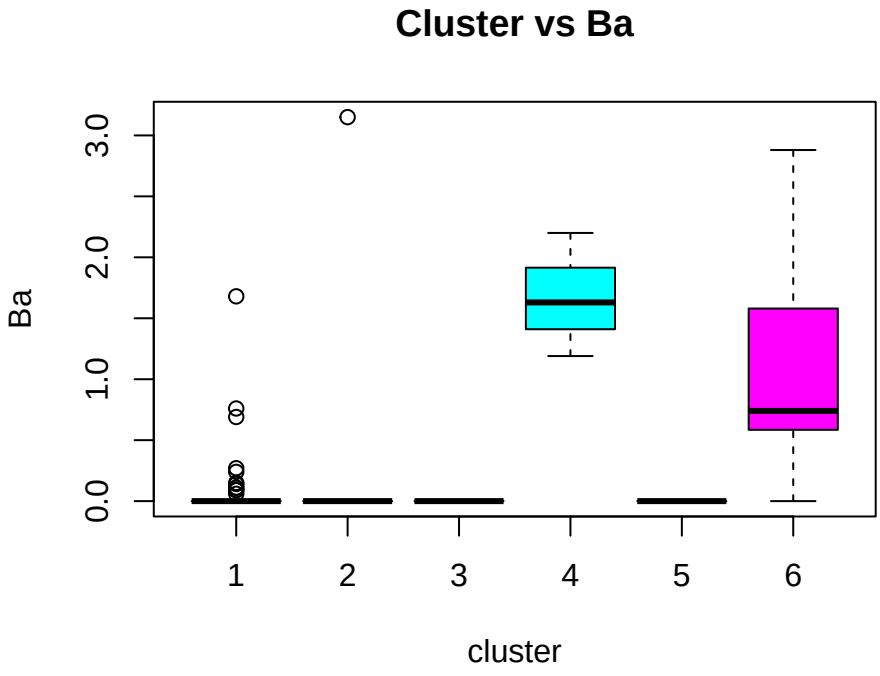
Rysunek 26: Wykresy pudełkowe dla różnych klastrów, dla cechy Si



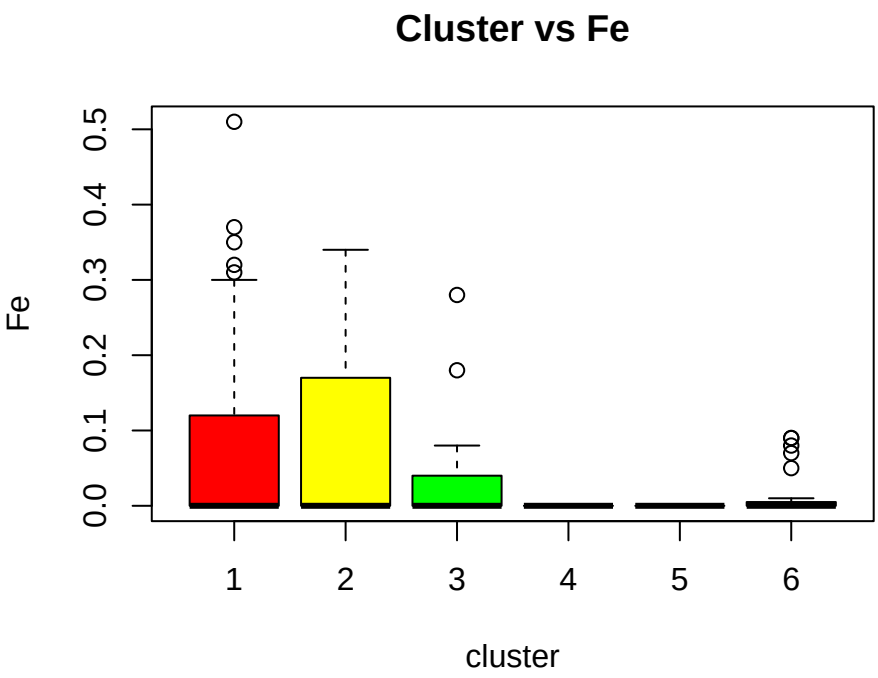
Rysunek 27: Wykresy pudełkowe dla różnych klastrów, dla cechy K



Rysunek 28: Wykresy pudełkowe dla różnych klastrów, dla cechy Ca



Rysunek 29: Wykresy pudełkowe dla różnych klastrów, dla cechy Ba



Rysunek 30: Wykresy pudełkowe dla różnych klastrów, dla cechy Fe

Dla tej metody można zaobserwować nieco większe zróżnicowanie cech niż dla PAM. Najbardziej wyróżniają się RI, Si (podobnie jak w PAM), ale też Ca.

3.5 Wnioski

Ze wszystkich tych metod najlepiej sprawdziła się metoda PAM, dająca niemal 60% poprawnych przypasowań dla 6 klastrów. Potwierdza to lekkie podobieństwo pomiędzy wykresami pudełkowymi dla klastrów i dla poszczególnych klas (sprawozdanie 3). Dla metody AGNES, choć udało się zgromadzić większe rozróżnienie w obrębie poszczególnych cech, to jednak poskutowało to gorszymi wynikami w przypisywaniu poprawnych klas.