Raport nr 4

Emilia Kowal [249716], Jakub Dworzański [249703]

14 czerwca 2020

Spis treści

1 Zaawansowane metody klasyfikacji

1.1 Krótki opis zagadnienia

W tym ćwiczeniu, będziemy kontynuowali analizę zbioru danych o odłamkach szkła.

```
data(Glass)
head(Glass)
```

Tabela 1: Przykładowe dane

RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0	0.00	1
1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0	0.00	1
1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0	0.00	1
1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0	0.00	1
1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0	0.00	1
1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0	0.26	1

1.2 Opis eksperymentów/analiz

Tym razem skorzystamy z bardziej zaawansownych metod. Rozpoczniemy od zastosowania rodzin klasyfikatorów. Skupimy się na algorytmach bagging, boosting oraz random forest.

Postaramy się odkryć, jak zadziałają te algorytmy. Ponadto, porównamy ich dokładność z poprzednimi wynikami. Sprawdzimy również, jak złożoność algorytmu wpływa na czas potrzebny do zbudowania modelu.

Następnie, wykorzystamy algorytm SVM - maszynę wektorów podpierających. Wykorzystując różne funkcje jądrowe oraz badając przestrzenie parametrów, postaramy się odkryć, czy ten algorytm obniży błąd klasyfikacji.

Tym razem, aby ocenić jakość predykcji, będziemy korzystać z 5-, sprawdzianu krzyżowego, metody *bootstrap* oraz *632+*.

Zastosujemy do tego następujące funkcje pomocnicze, odpowiadające za uczenie modelu oraz estymację błędów.

```
mypredict <- function(model, newdata) predict(model, newdata, type="class")
adaboost.predict <- function(model, newdata){
  return(as.factor(predict(model, newdata)$class))
}
nasz.errorest <- function(formula, data, model, predict=mypredict, ...){
  cv.5 <- errorest(
    formula=formula, data=data, model=model, predict=predict,
    estimator="cv", est.para=control.errorest(k = 5), ...
  )$error
  bootstrap <- errorest(
    formula=formula, data=data, model=model, predict=predict,
    estimator="boot", est.para=control.errorest(nboot = 5), ...
  )$error
  err632plus <- errorest(
    formula=formula, data=data, model=model, predict=predict,
    estimator="boot", est.para=control.errorest(nboot = 5), ...
  )$error
  return(data.frame(
    paste0(
      round(100*c(cv.5, bootstrap, err632plus), 2),
      "%"
    ),
    row.names=c(
      "CV5",
      "bootstrap (5 powtórzeń)",
      "632+ (5 powtórzeń)"
    )
  ))
}
wyniki <- data.frame(row.names=c(
      "CV5",
      "bootstrap (5 powtórzeń)",
      "632+ (5 powtórzeń)"
    ))
```

Tabela 2: Ważność zmiennych na podstawie konstrukcji drzewa klasyfikacyjnego

	Ważność zmiennej
RI	46.621949
Al	41.239493
Ca	39.191590
Mg	38.944662
Na	27.423009
Ba	26.419293
K	23.641254
Si	21.378069
Fe	8.000758

Tabela 3: Najlepsze wyniki dla drzewa klasyfikacyjnego, otrzymane podczas pierwszej części analizy.

	Błąd klasyfikacji
CV5	30.84%
bootstrap (5 powtórzeń)	35.43%
632+ (5 powtórzeń)	35.03%

1.3 Rodziny klasyfikatorów

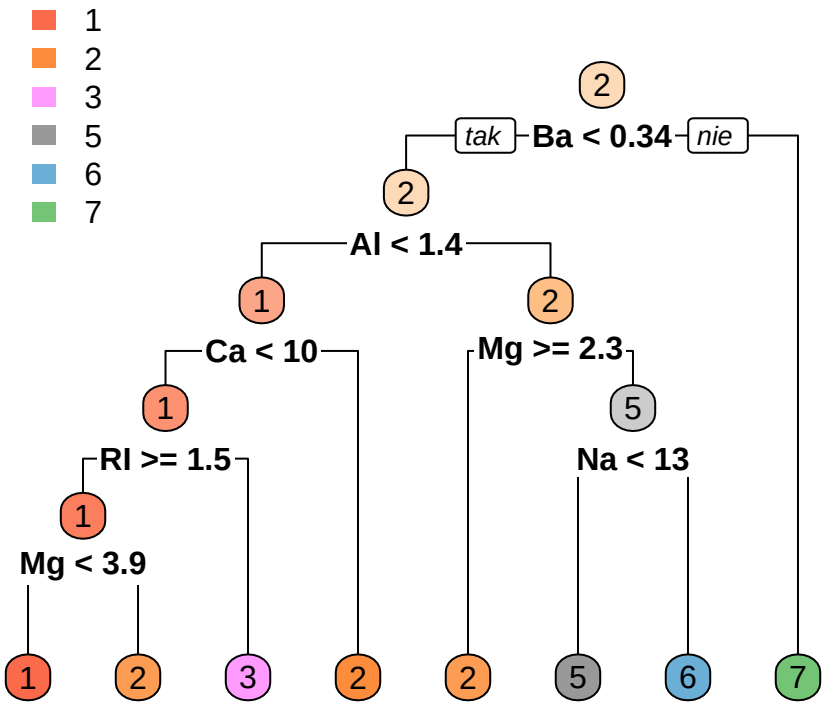
Aby mieć punkt odniesienia podczas oceny wyników, skorzystamy z drzewa decyzyjnego, które będzie wykorzystywane jako klasyfikator bazowy w tym ćwiczeniu.

Przypomnijmy, że w pierwszej części analizy, najlepszy rezultat osiągneliśmy poprzez tworzenie drzewa na podstawie 6 najważniejszych zmiennych oraz przycinanie go po konstrukcji za pomocą metody jednego odchylenia.

```
drzewo <- rpart(Type~., Glass, minsplit=1)
waznosc.drzewo <- drzewo$variable.importance
waznosc.drzewo
```

```
nasze.drzewo <- function(...){
  drzewo <- rpart(..., minsplit=1)
  cptable <- data.frame(drzewo$cptable)
  xerror.min <- min(cptable$xerror)
  xstd.min <- max(cptable[which(cptable$xerror == xerror.min),]$xstd)
  ponizej.1.sd <- cptable[which(cptable$xerror < xerror.min + xstd.min),]$CP
  return(prune(drzewo, cp=max(ponizej.1.sd)))
}
wyniki["drzewo"] <- nasz.errorest(
  Type ~ RI + Al + Ca + Mg + Na + Ba, Glass, nasze.drzewo
)
wyniki["drzewo"]
```

```
drzewo <- nasze.drzewo(Type ~ RI + Al + Ca + Mg + Na + Ba, Glass)
rpart.plot(drzewo)
```



Rysunek 1: Przykładowe drzewo klasyfikacyjne

Na rys. 1 możemy zobaczyć przykładowe drzewo decyzyjne, stworzone tą metodą na podstawie całego zbioru danych.

Po zapoznaniu się z dotychczasowymi wynikami, sprawdzimy, jak z klasyfikacją poradzą sobie komitety klasyfikatorów.

```
wyniki["bagging"] <- nasz.errorest(
  Type ~ ., Glass, ipred::bagging
)
wyniki["boosting"] <- nasz.errorest(
  Type~., Glass, adabag::boosting, predict=adaboost.predict
)
wyniki["random.forest"] <- nasz.errorest(
  Type~., Glass, randomForest
)
```

W tabeli 4 widzimy, że zastosowanie algorytmów typu *ensemble*, pozwala na znaczną poprawę wyników (względem klasyfikatora bazowego - drzewa klasyfikacyjnego) nawet dla komitetów z domyślnymi parametrami.

Ponadto, różnice błędu klasyfikacji pomiędzy poszczególnymi komitetami, są stosunkowo niewielkie (rzędu 2-3 punktów procentowych).

Niestety, wraz ze wzrostem dokładności, zdecydowanie wzrasta również czas, potrzebny na trening komitetu klasyfikatorów. Sprawdzimy teraz, jak bardzo wydłuża się czas potrzebny do zbudowania poszczególnych modeli.

Tabela 4: Porównanie błędów klasyfikacji.

	drzewo	bagging	random forest	boosting
CV5	30.84%	25.23%	23.83%	21.5%
bootstrap (5 powtórzeń)	35.43%	26.15%	23.49%	25.3%
632+ (5 powtórzeń)	35.03%	26.02%	24.57%	27.76%

Tabela 5: Czas konstrukcji względem czasu konstrukcji drzewa decyzyjnego.

	czas
drzewo	100%
las losowy	587.57%
bagging	619.79%
boosting	156064.08%

```
czas<- data.frame(row.names=c("czas"))

start <- Sys.time()
rpart(Type~., Glass)
stop <- Sys.time()
czas.drzewo <- as.numeric(stop-start)
czas["drzewo"] <- 1

start <- Sys.time()
randomForest(Type~., Glass)
stop <- Sys.time()
czas["random.forest"] <- as.numeric(stop-start) / czas.drzewo

start <- Sys.time()
ipred::bagging(Type~., Glass)
stop <- Sys.time()
czas["bagging"] <- as.numeric(stop-start) / czas.drzewo

start <- Sys.time()
boosting(Type~., Glass)
stop <- Sys.time()
czas["boosting"] <- as.numeric(stop-start) / czas.drzewo

czas
```

W tabeli 5 widzimy, że faktycznie czas potrzebny na zbudowanie komitetu klasyfikatorów może być istotnie dłuższy niż czas potrzebny na zbudowanie drzewa.

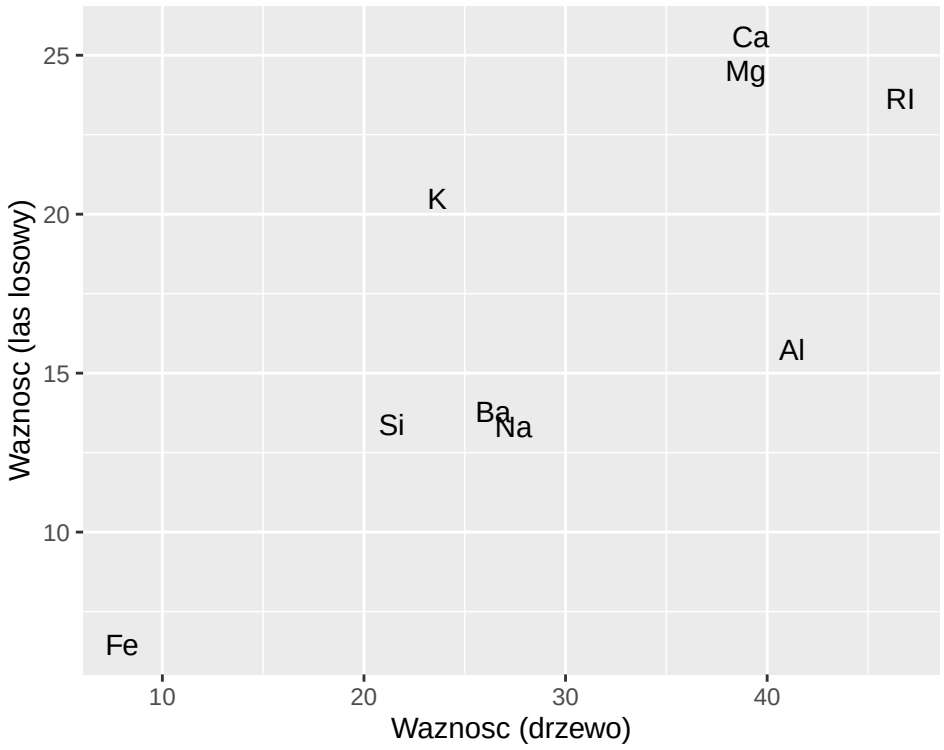
Korzystając z konstrukcji lasu losowego, możemy też wyznaczyć ważność cech, podobnie jak dla pojedynczego drzewa klasyfikacyjnego.

```
model.rf <- randomForest(Type~., Glass)
model.rf$importance
```

Tabela 6: Ważność zmiennych na podstawie konstrukcji lasu losowego.

	Ważność zmiennej
RI	23.637974
Na	15.738473
Mg	25.599501
Al	24.533284
Si	13.314850
K	13.800157
Ca	20.503917
Ba	13.394118
Fe	6.479918

```
porownanie <- data.frame(
  waznosc.drzewo,
  model.rf$importance
)
names(porownanie) <- c("drzewo", "las.losowy")
ggplot(
  porownanie,
  aes(x=drzewo, y=las.losowy, label=rownames(porownanie))
) + geom_text()
```



Rysunek 2: Porównanie ważności zmiennych

Tabela 7: Porównanie SVM z wykorzystaniem domyślnych parametrów i różnych funkcji jądrowych.

	liniowe	wielomianowe	radialne	sigmoidalne
CV5	37.38%	51.87%	31.31%	44.86%
bootstrap (15 powtórzeń)	38.36%	49.96%	32.76%	50.87%
632+ (15 powtórzeń)	34.71%	45.5%	28.72%	47.77%

W tabeli 6 widzimy ważnośćm przypisaną poszczególnym cechom, na podstawie konstrukcji lasu losowego. Natomiast na wykresie 2 możemy zobaczyć, że istotność cech otrzymana na podstawie drzewa klasyfikacyjnego częściowo pokrywa się z istotnością na podstawie lasu losowego. W szczególności, zbiór 5 najważniejszych cech pokrywa się ze sobą dla obu "rankingów". Ponadto, oba sposoby wskazały, że zawartość żelaza jest najmniej istotną z cech.

1.4 Metoda wektorów nośnych (SVM)

Analizę algorytmu SVM rozpoczniemy od sprawdzenia błędu klasyfikacji dla domyślnych parametrów, korzystając z różnych funkcji jądrowych.

```
wyniki["linear SVM"] <- nasz.errorest(Type~., Glass, svm, kernel="linear")
wyniki["polynomial SVM"] <- nasz.errorest(Type~., Glass, svm, kernel="polynomial")
wyniki["radial SVM"] <- nasz.errorest(Type~., Glass, svm, kernel="radial")

wyniki
```

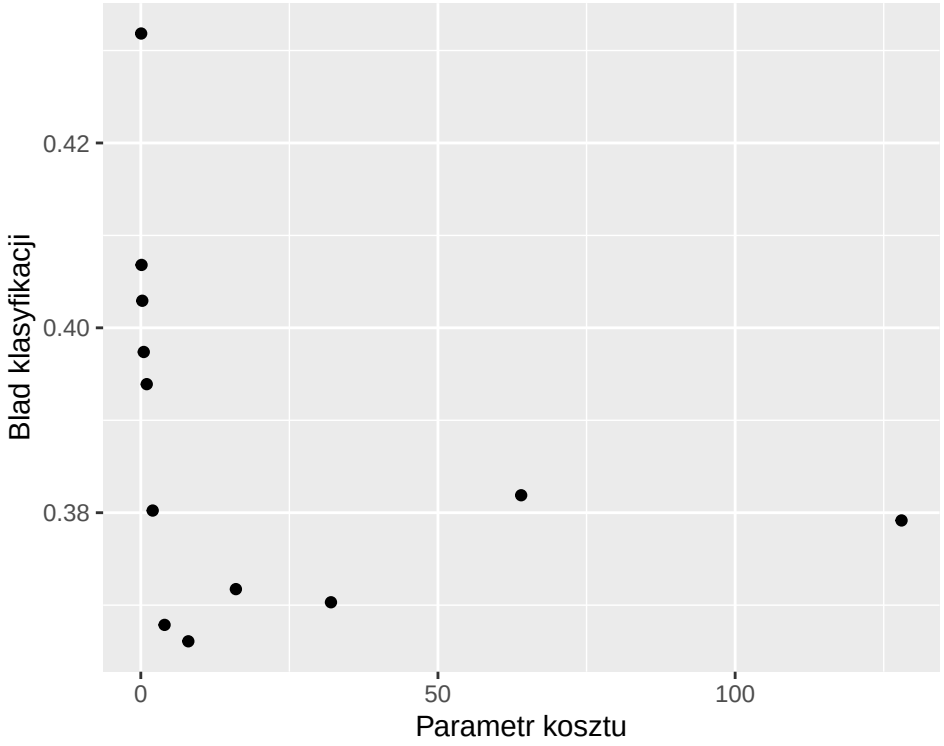
W tabeli 7 widzimy, że dla domyślnych parametrów, najlepiej radzą sobie SVM z liniową funkcją jądrową oraz SVM z radialną funkcją jądrową. W tej samej tabeli, widzimy również, że wybór funkcji jądrowej w sposób znaczny wpływa na dokładność klasyfikatora. Możemy jednak przypuszczać, że ma to również związek z większą wrażliwością na dobór parametrów dla SVM z wielomianową lub sigmoidalną funkcją jądrową.

Postaramy się teraz sprawdzić, jaki wpływ na klasyfikację, przy pomocy SVM z jądrem liniowym, ma parametr kosztu C.

```
svm.tuning <- tune.svm(
  Type~., data=Glass, kernel="linear", cost=2^(-4:7),
  tunecontrol=tune.control(sampling="boot", nboot=25)
)
ggplot(svm.tuning$performances, aes(x=cost, y=error)) + geom_point()
svm.tuning$best.parameters
svm.tuning$best.performance
```

Tabela 8: Parametry klasyfikacji dla SVM z jądrem liniowym, minimalizujące błąd klasyfikacji.

Parametr kosztu	Błąd klasyfikacji
8	36.61%



Rysunek 3: Wykres błędu klasyfikacji w zależności od parametru kosztu dla SVM z liniową funkcją jądrową.

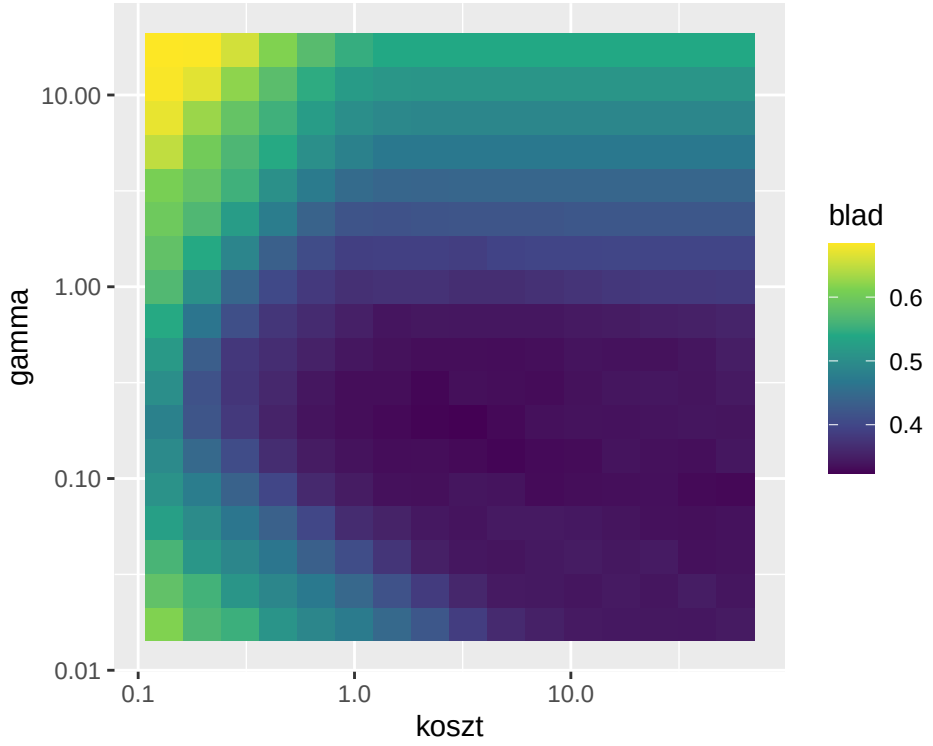
Na rysunku 3 widzimy, że błąd klasyfikacji zmienia się w zależności od parametru kosztu. Spośród zbadanych przez nas parametrów, estymowany błąd klasyfikacji osiąga minimum dla $C = 8$‘. Otrzymujemy wtedy wynik 36.61% (tab. 8).

Widzimy, że nie doszło do znacznego zwiększenia dokładności klasyfikacji względem domyślnych parametrów. Dlatego też, spróbujemy teraz ”dostroić” parametry dla SVM z radialną funkcją jądrową, ponieważ może być on bardziej wrażliwy na dobór parametrów.

```
svm.tuning <- tune.svm(
  Type~., data=Glass, kernel="radial",
  cost=1.5^(-5:10), gamma=1.5^(-10:7),
  tunecontrol=tune.control(sampling="boot", nboot=25)
)
ggplot(svm.tuning$performances, aes(x=cost, y=gamma, fill=error)) +
  geom_tile() + scale_y_log10() + scale_x_log10()
svm.tuning$best.parameters
svm.tuning$best.performance
```

Tabela 9: Parametry klasyfikacji dla SVM z jądrem radialnym, minimalizujące błąd klasyfikacji.

Parametr gamma	Parametr kosztu	Błąd klasyfikacji
0.1975309	3.375	32.38%



Rysunek 4: Wykres błędu klasyfikacji dla SVM z jądrem radialnym w zależności od parametrów kosztu i gamma.

Na wykresie 4 widzimy, że dobór parametrów może mieć istotny wpływ na dokładność klasyfikacji. Mimo tego, nie udało nam się znacznie poprawić dokładności względem klasyfikatora z domyślnymi parametrami (tab. 9).

1.5 Wnioski

Na podstawie przeprowadzonych eksperymentów, możemy stwierdzić, że bardziej zaawansowane metody klasyfikacji pozwalają na osiągnięcie dużo niższego błędu klasyfikacji. W przypadku komitetów klasyfikatorów, osiągaliśmy obniżenie błędu klasyfikacji o niemalże 50%.

Widzieliśmy również, że wyższa skuteczność idzie w parze z większą złożonością algorytmu, przez co budowanie modeli może się znacznie wydłużyć.

W przypadku SVM, poprawa wyników nie była tak znacząca. Ten problem może być spowodowany przez różne czynniki. Wpływ na wysokość błędu predykcji może mieć to, że SVM jest algorytmem do klasyfikacji binarnej, a wykorzystywanie go do problemów wieloklasowych, może pogarszać jego przewagę nad pozostałymi metodami. Ponadto, problemem może być odnalezienie lokalnego minimum w przestrzeni parametrów modelu. Mimo wyników porównywalnych z wcześniej poznanymi klasyfikatorami, mogliśmy się przekonać, że w przypadku SVM dobór parametrów jest bardzo istotny i ma duży wpływ na błąd klasyfikacji.

Tabela 10: Przykładowe dane

RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0	0.00	1
1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0	0.00	1
1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0	0.00	1
1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0	0.00	1
1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0	0.00	1
1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0	0.26	1

2 Analiza skupień - algorytmy grupujące i hierarchiczne

2.1 Krótki opis zagadnienia

W tej sekcji będziemy zajmować się analizą skupień z wykorzystaniem algorytmów grupujących oraz hierarchicznych. Analizy dokonujemy na zbiorze Glass z biblioteki datasets.

```
data(Glass)
head(Glass)
```

Dane (tabela: 1) składają się z informacji, dotyczących 214 odłamków. Każdy z rekordów składa się ze współczynnika załamania światła (`RI`) oraz stężenia różnych pierwiastków (w odpowiadających tlenkach) w badanych próbkach. Oprócz tego, posiadamy informacje o typie szkła, z którego odłamkiem mamy do czynienia. Razem, mamy 7 takich klas, z których tylko 6 występuje w danych. Łącznie, dla każdej próbki występuje 10 cech. Ponadto, w zbiorze nie ma wartości brakujących.

2.2 Opis eksperymentów/analiz

- W pierwszym kroku zastosujemy algorytm grupujący PAM oraz algorytm hierarchiczny AGNES.
- Następnie zobrazujemy wyniki algorytmów grupujących wykorzystując wykresy rozrzutu. Aby przedstawić rezultaty na wykresie dwuwymiarowym, zastosujemy algorytm PCA.
- Dla algorytmów hierarchicznych porównamy dendrogramy dla różnych metod łączenia skupień.
- Ocenimy jakość grupowania z wykorzystaniem wskaźników zewnętrznych, takich jak macierz kontyngencji oraz wskaźników zewnętrznych, np. indeksu silhouette.
- Następnie postaramy się dobrać optymalną liczbę klas dla zbioru danych Glass.

2.3 Wyniki

Usuwamy ze zbioru Glass zmienną grupującą Type, zawierającą etykietki klas.

```
glass.cechy <- Glass[, -10]
glass.real.etykiety <- Glass[, 10]
```

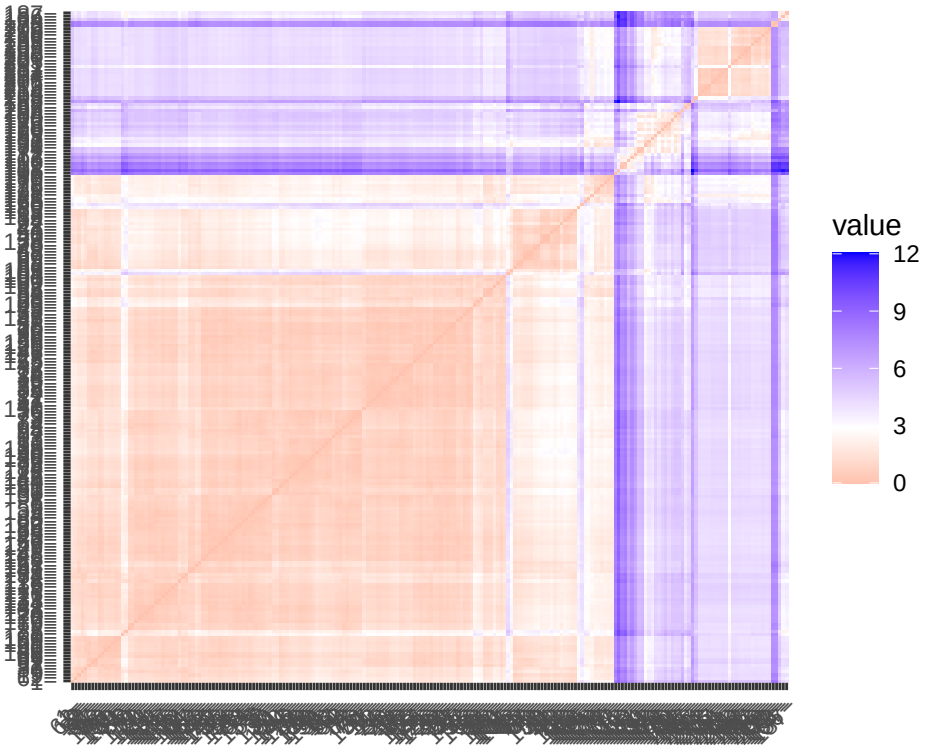
Tabela 11: Medoidy reprezentujące klastry dla k=6

66	148	28	171	198	172

Zbiór Glass zawiera 214 wierszy, zatem nie ma konieczności losowania podzbioru w celu zredukowania czasu działania algorytmów.
Przed wyznaczeniem macierzy niepodobieństw nie standaryzujemy danych. Podczas analizy chemicznej, do rozstrzygnięcia o typie szkła, istotnym jest, który z pierwiastków dominuje w składzie. Następnie wyznaczamy macierz niepodobieństw z metryką euklidesową dla nieustandaryzowanych danych.

```
diss.mtrx.glass <- daisy(glass.cechy)
```

Wizualizacja macierzy odmienności po uporządkowaniu dla cech ze zbioru *Glass*:



Rysunek 5: Macierz odmienności po uporządkowaniu

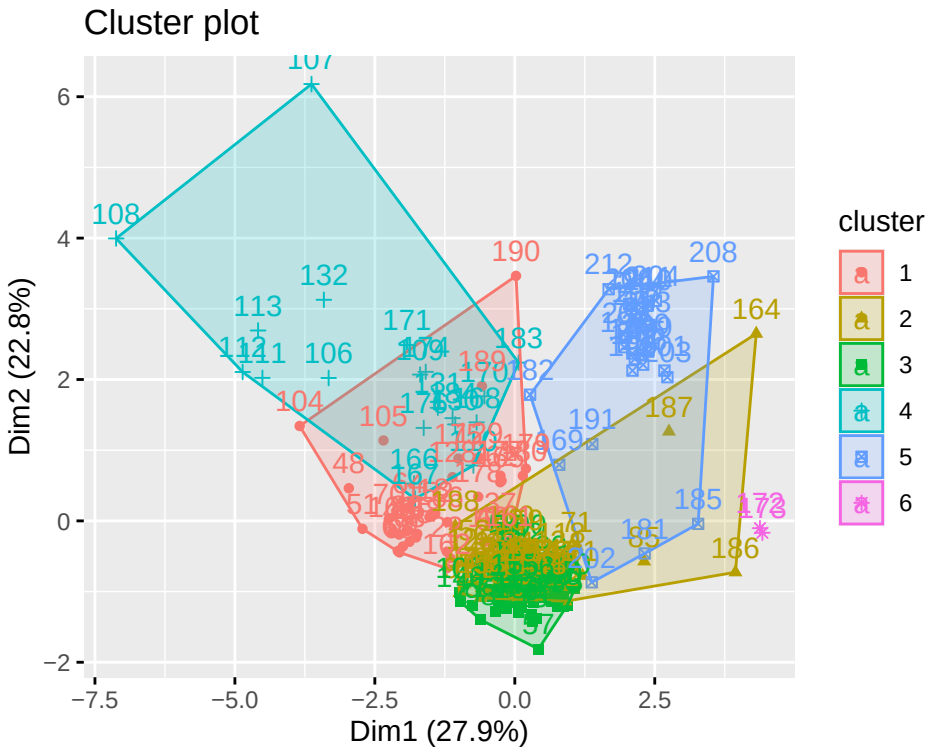
2.3.1 Algorytm PAM (Partitioning Around Medoids)

W pierwszej kolejności do analizy skupień wykorzystamy algorytm grupujący PAM, przyjmując liczbę klas k=6 zgodną z rzeczywistymi etykietkami klas.

```
glass.pam <- pam(x=diss.mtrx.glass, diss=TRUE, k=6)
```

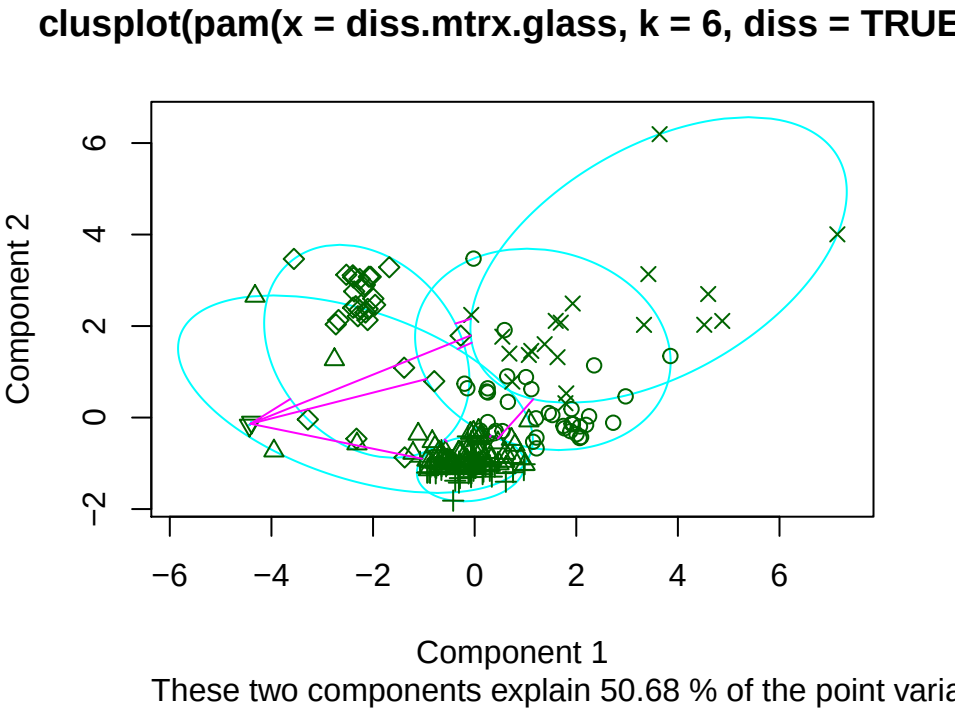
W tabeli 11 widzimy medoidy reprezentujące klastry dla 6 klas.

Tworzymy wizualizację z wykorzystaniem algorytmu PCA za pomocą funkcji *fviz_cluster* z biblioteki *factoextra*.



Rysunek 6: Wizualizacja PAM z wykorzystaniem algorytmu PCA

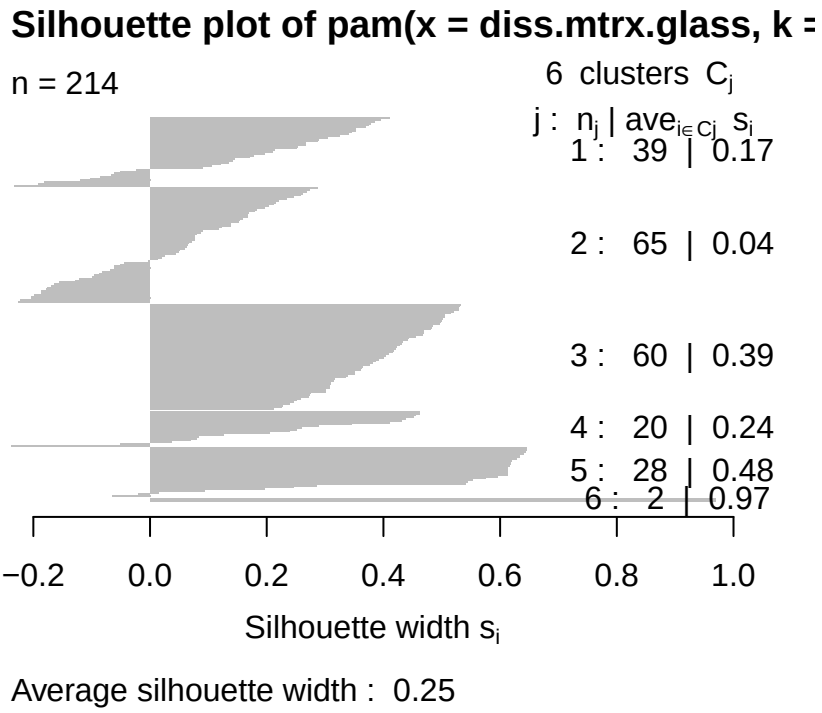
Skupiska nie są dobrze odseparowane.
Skuteczność algorytmu PAM dla zbioru danych *Glass* sprawdzimy również wizualizując wynik wskaźnika *silhouette*.



Rysunek 7: Wskaźnik silhouette dla algorytmu PAM

Tabela 12: Tabela kontyngencji dla PAM i k=6

1	2	3	5	6	7
22	4	5	2	4	2
17	35	9	1	0	3
31	26	3	0	0	0
0	11	0	7	2	0
0	0	0	1	3	24
0	0	0	2	0	0



Rysunek 8: Wskaźnik silhouette dla algorytmu PAM

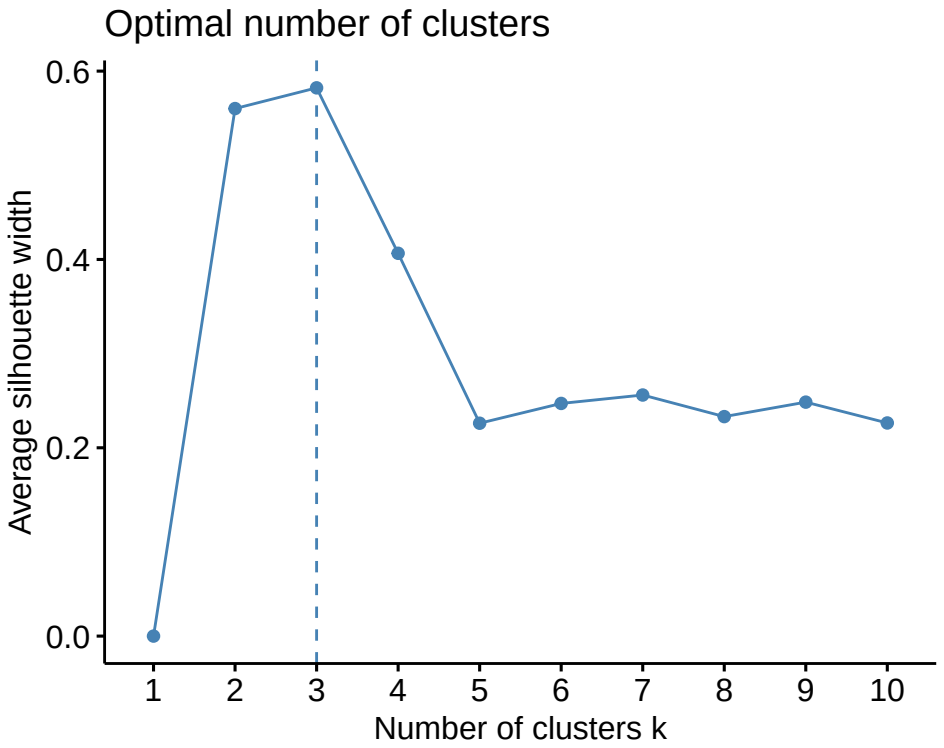
Zbadamy również poziom zgodności otrzymanych wyników z rzeczywistymi etykietami obiektów wykorzystując tabelę kontyngencji. Zgodnie z tabelą 12 dostajemy zgodność ok. 48

Dla porównania wyznaczyliśmy również macierz kontyngencji dla danych ustandaryzowanych i otrzymaliśmy zgodność na poziomie ok. 43% (tabela: **??**), zatem o 5 punktów procentowych gorszą od wyniku dla danych nieustandaryzowanych. W związku z tym w dalszej części raportu pracujemy już tylko na zbiorze nieskalowanym.

Następnie wyznaczyliśmy sugerowaną liczbę klas z pomocą algorytmu *fviz_nbclust* w oparciu o wskaźnik *silhouette*.

Tabela 13: Tabela kontyngencji dla PAM na danych ustandaryzowanych i k=6

1	2	3	5	6	7
17	2	2	0	0	3
40	43	12	2	4	3
13	21	3	2	0	0
0	10	0	6	2	0
0	0	0	1	3	23
0	0	0	2	0	0



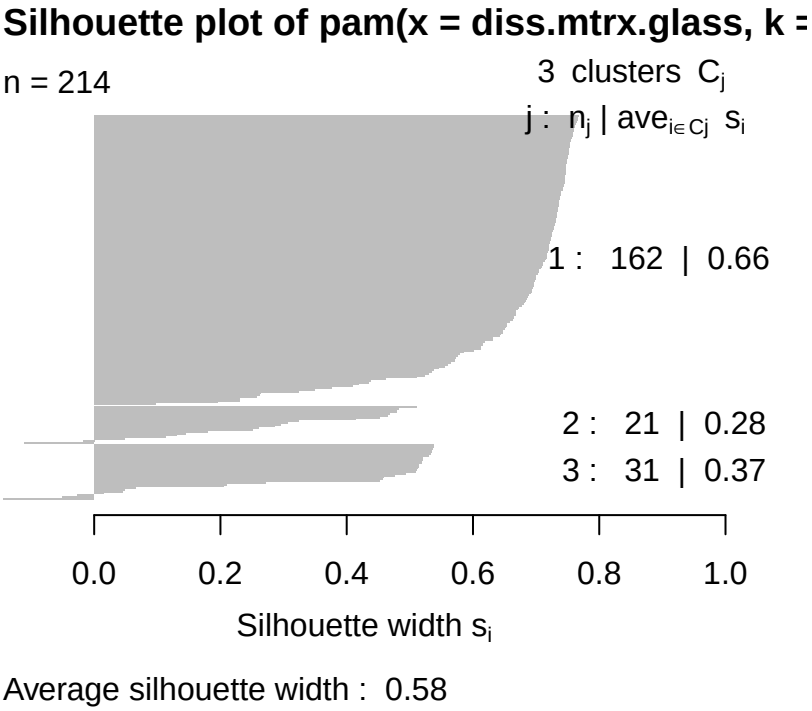
Rysunek 9: Sugerowana liczba klas dla algorytmu PAM

Dostajemy informację, iż optymalna liczba klas wynosi 3. Wnioskujemy, że niektóre typy szkła mogą mieć zbliżony skład, a w konsekwencji zostają przypisane do skupienia, gdzie poszczególne wyniki mają podobne właściwości.

Powtarzamy analizę z wykorzystaniem algorytmu PAM tym razem dla 3 klas.

Tabela 14: Tabela kontyngencji dla PAM i k=3

1	2	3	5	6	7
70	64	17	3	4	4
0	12	0	7	2	0
0	0	0	3	3	25



Rysunek 10: PAM bez standaryzacji dla 3 klas.

Otrzymujemy następującą tabelę kontyngencji: 14 oraz poziom zgodności 50%.

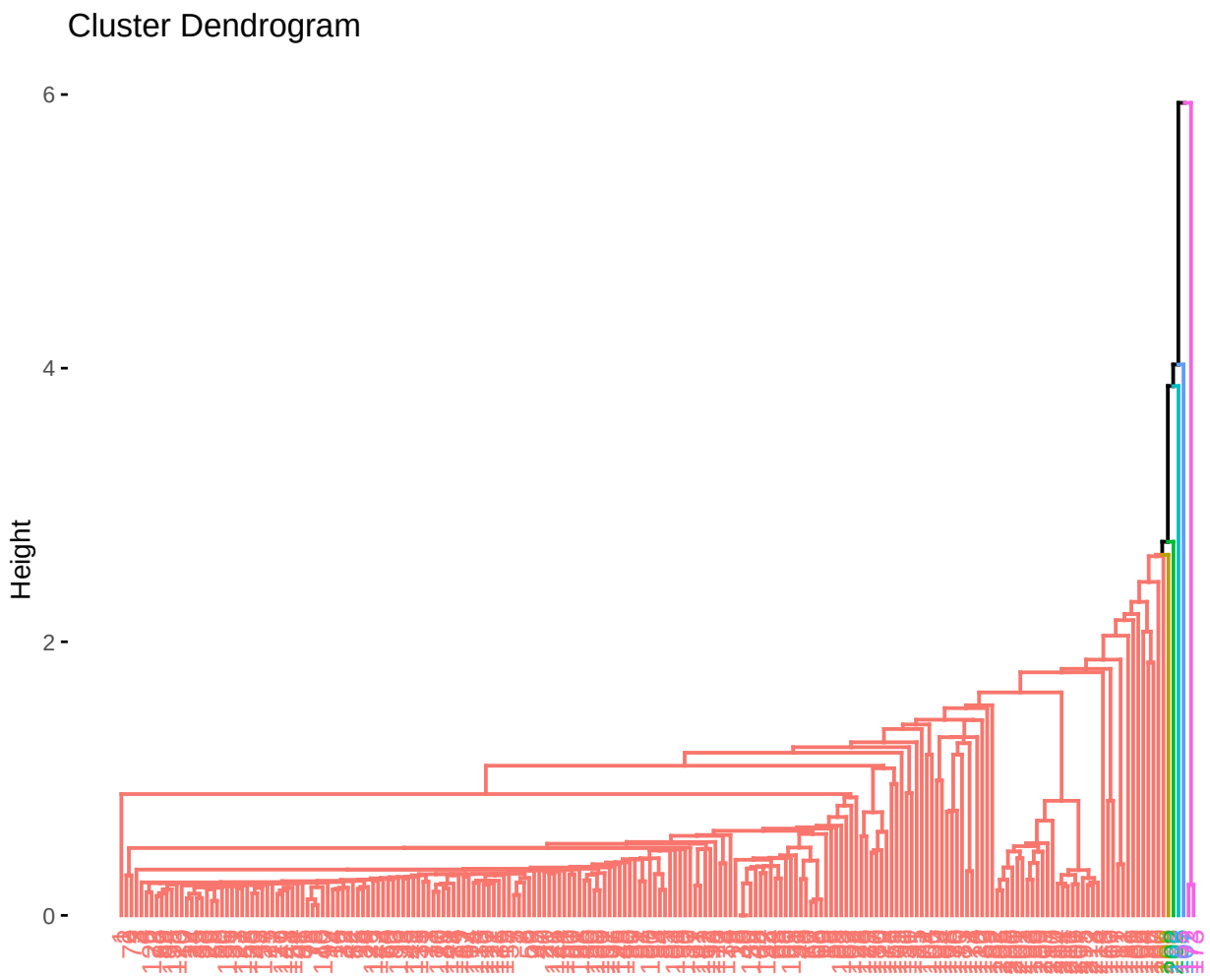
```
## Cases in matched pairs: 50 %
##   1   2   3
## "1" "2" "7"
```

2.3.2 Algorytm AGNES (Aglomerative Nesting)

W tej sekcji do analizy skupień wykorzystamy algorytm hierarchiczny AGNES. Wyniki działania przeanalizujemy dla różnych metod łączenia skupień. Na początku przyjmujemy rzeczywistą liczbę klas równą 6.

```
glass.agnes.single <- agnes(x=diss.mtrx.glass, diss=TRUE, method="single")
```

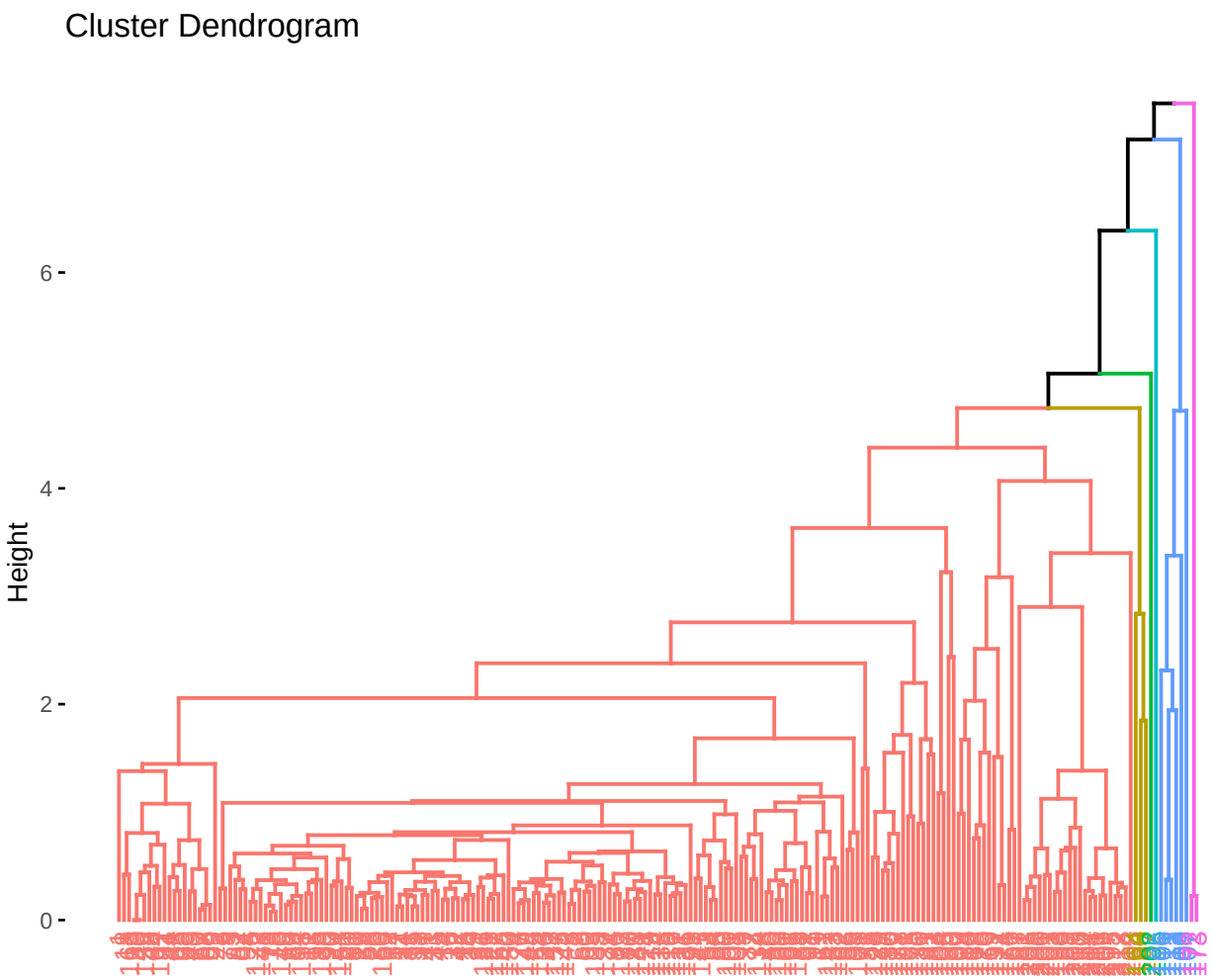
Dendrogram dla algorytmu AGNES z metodą *single*.



Rysunek 11: AGNES z metodą single dla k=6

```
glass.agnes.avg <- agnes(x=diss.mtrx.glass, diss=TRUE, method="average")
```

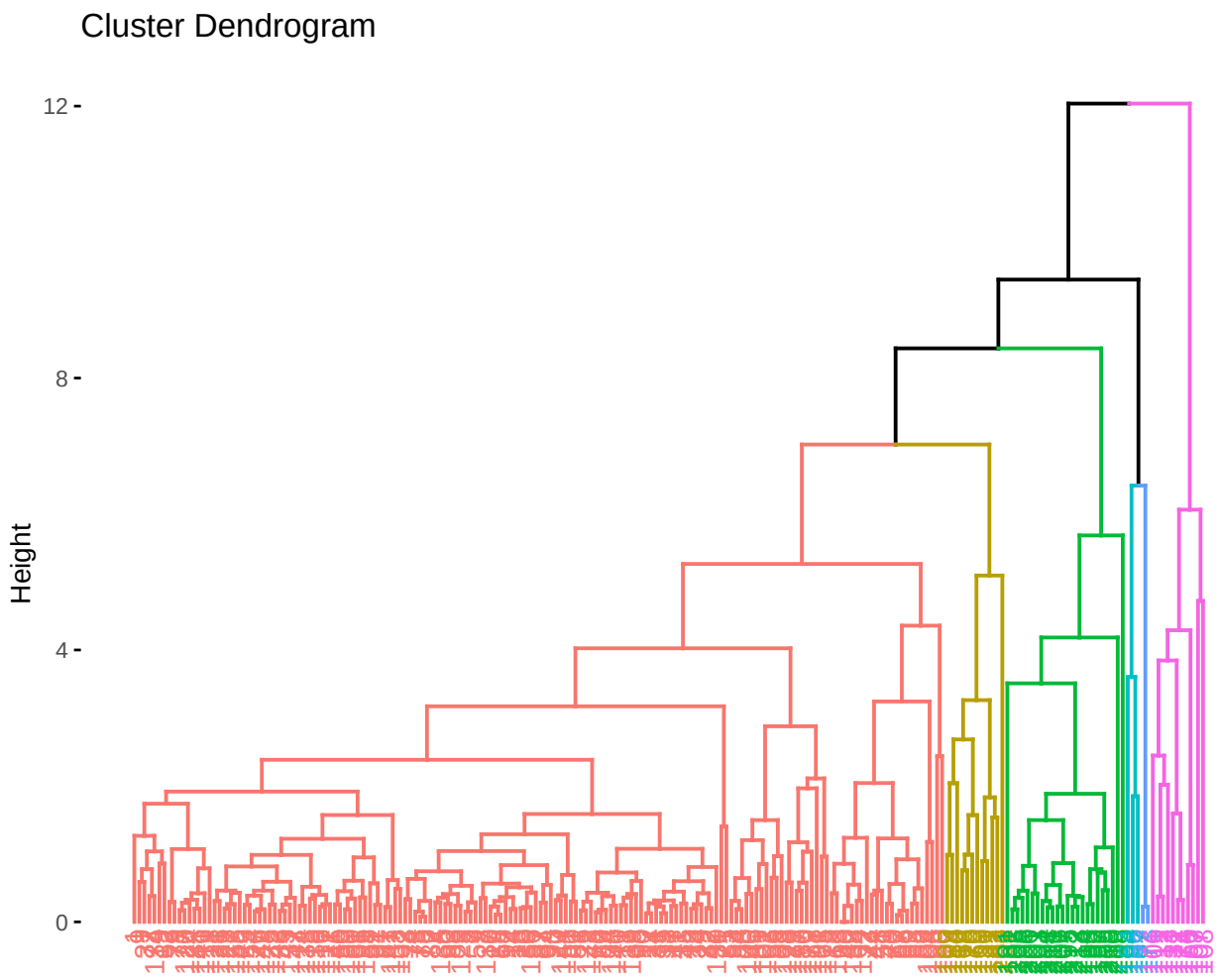
Dendrogram dla algorytmu AGNES z metodą *average*.



Rysunek 12: AGNES z metodą average dla k=6

```
glass.agnes.complete <- agnes(x=diss.mtrx.glass, diss=TRUE, method="complete")
```

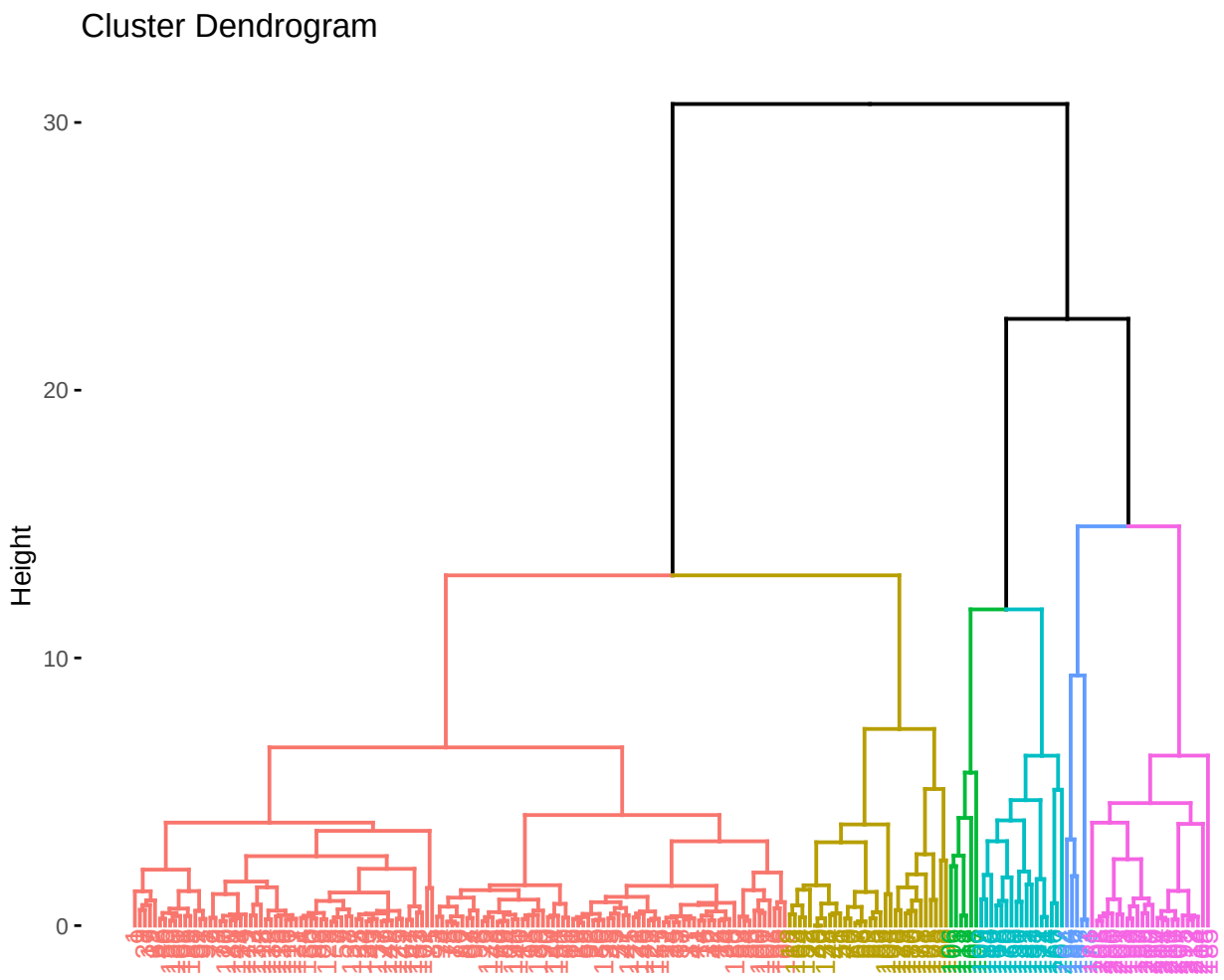
Dendrogram dla algorytmu AGNES z metodą *complete*.



Rysunek 13: AGNES z metodą complete dla k=6

```
glass.agnes.ward <- agnes(x=diss.mtrx.glass, diss=TRUE, method="ward")
```

Dendrogram dla algorytmu AGNES z metodą *Warda*.



Rysunek 14: AGNES z metodą Warda dla k=6

Na podstawie samych wykresów można wywnioskować, że najlepszy podział na skupienia wyznacza metoda *Warda* oraz *complete*.
Hipotezę sprawdzimy wykorzystując wskaźnik wewnętrzny *silhouette* i tabel kontyngencji oraz porównując wyniki wskaźników dla każdej z metod łączenia skupień.

```
cut.glass.agnes.sing <- cutree(glass.agnes.single, k=6)
sil.glass.agnes.sing <- silhouette(cut.glass.agnes.sing, dist=diss.mtrx.glass)
 fviz_silhouette(sil.glass.agnes.sing)

##   cluster size ave.sil.width
## 1       1  208          0.45
## 2       2    1          0.00
## 3       3    1          0.00
## 4       4    2          0.97
## 5       5    1          0.00
## 6       6    1          0.00
```

Tabela 15: Tabela kontyngencji dla metody single

1	2	3	5	6	7
70	74	17	11	8	28
0	1	0	0	0	0
0	1	0	0	0	0
0	0	0	2	0	0
0	0	0	0	1	0
0	0	0	0	0	1

Rysunek 15: Wskaźnik silhouette dla metody single

Macierz kontyngencji dla metody łączenia skupień *single*: 15. Otrzymujemy zgodność na poziomie: 37,38%.

```
##   cluster size ave.sil.width
## 1       1  201          0.44
## 2       2    6          0.54
## 3       3    3          0.47
## 4       4    2          0.96
## 5       5    1          0.00
## 6       6    1          0.00
```

Tabela 16: Tabela kontyngencji dla metody average

1	2	3	5	6	7
70	70	17	10	8	26
0	6	0	0	0	0
0	0	0	1	0	2
0	0	0	2	0	0
0	0	0	0	1	0
0	0	0	0	0	1

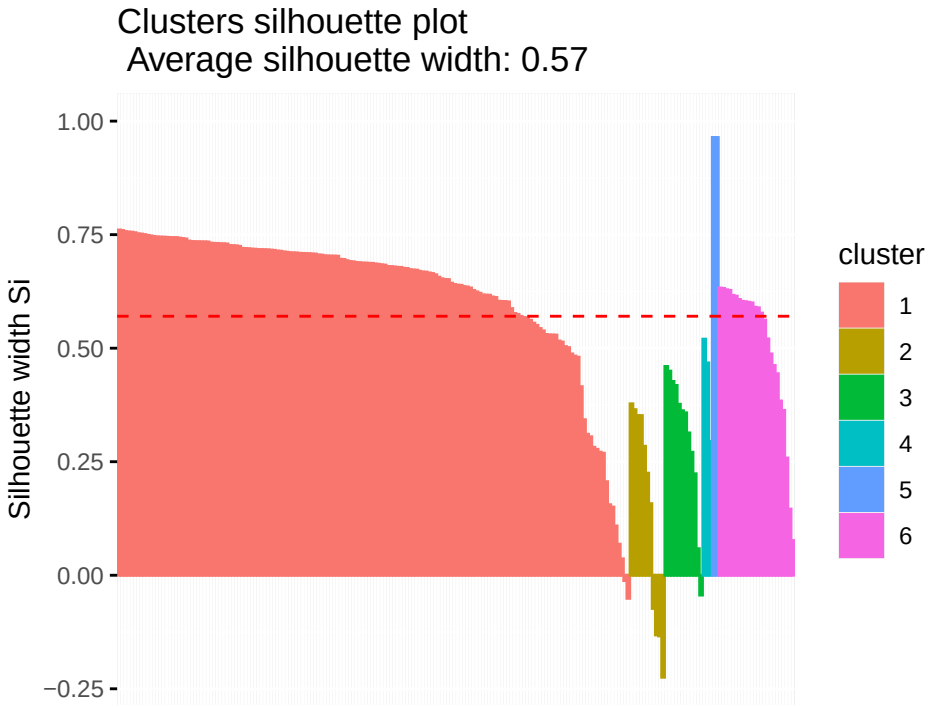
Rysunek 16: Wskaźnik silhouette dla metody average

Macierz kontyngencji dla metody łączenia skupień *average*: 16. Otrzymujemy zgodność na poziomie: 38.32 %.

```
##   cluster size ave.sil.width
## 1       1  162          0.63
## 2       2   11          0.14
## 3       3   12          0.31
## 4       4    3          0.43
## 5       5    2          0.96
## 6       6   24          0.51
```

Tabela 17: Tabela kontyngencji dla metody complete

1	2	3	5	6	7
70	65	17	2	4	4
0	7	0	4	0	0
0	4	0	4	3	1
0	0	0	1	0	2
0	0	0	2	0	0
0	0	0	0	2	22



Rysunek 17: Wskaźnik silhouette dla metody complete

Macierz kontyngencji dla metody łączenia skupień *complete*: 17. Otrzymujemy zgodność na poziomie: 50%.

```
##   cluster size ave.sil.width
## 1       1  130          0.56
## 2       2   32          0.20
## 3       3    6          0.32
## 4       4   17          0.22
## 5       5    5          0.12
## 6       6   24          0.56
```

Tabela 18: Tabela kontyngencji dla metody Warda

1	2	3	5	6	7
54	61	14	0	0	1
16	4	3	2	4	3
0	6	0	0	0	0
0	5	0	8	3	1
0	0	0	3	0	2
0	0	0	0	2	22

Rysunek 18: Wskaźnik silhouette dla metody Warda

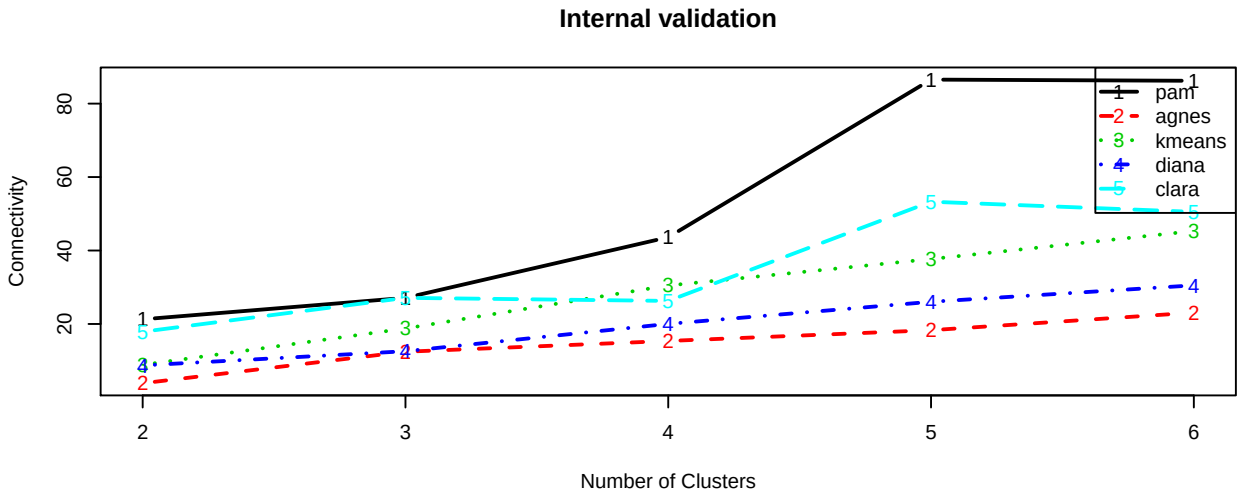
Macierz kontyngencji dla metody łączenia skupień *Warda*: 18. Otrzymujemy zgodność na poziomie: 54.21 %.

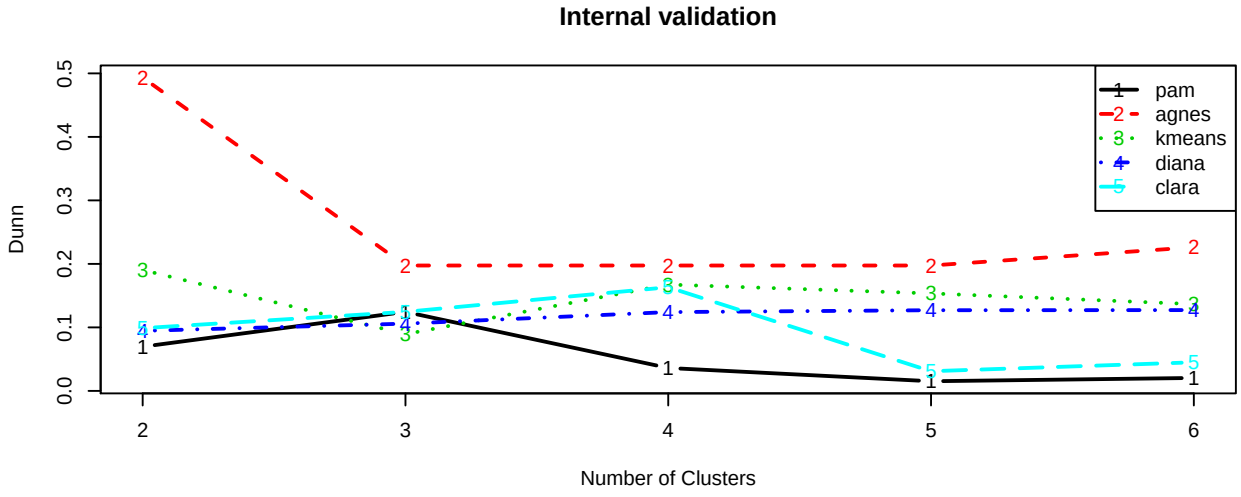
Na podstawie powyższej analizy zauważamy, że dla $k = 6$ metoda *Warda* uzyskuje osiąga lepsze wyniki dla wskaźnika zewnętrznego, natomiast metoda *complete* powoduje, że klastry są bardziej zwarte i lepiej od siebie odseparowane.

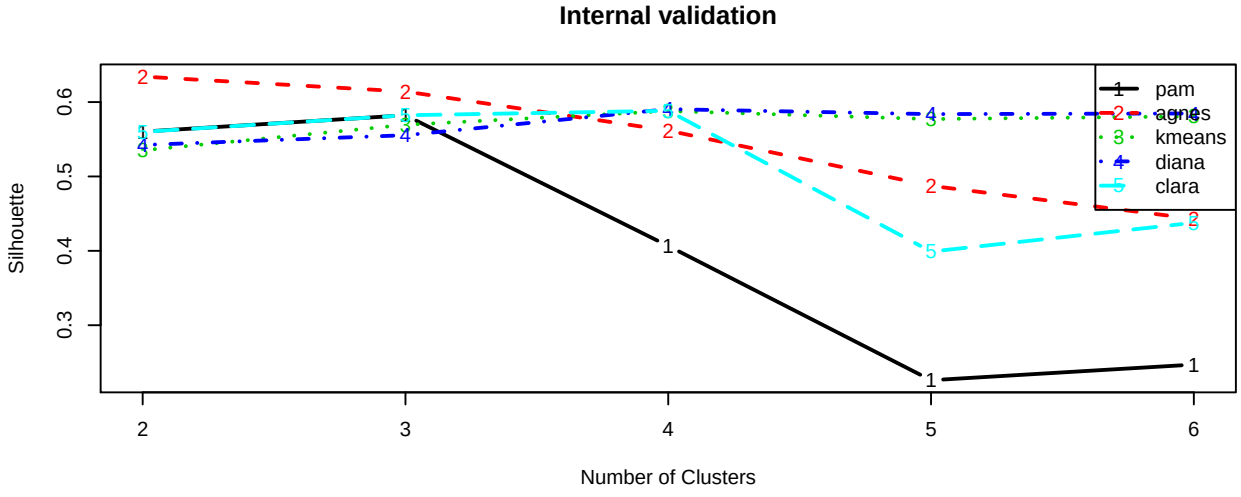
W celu dokładniejszej oceny jakości grupowania wykorzystamy bibliotekę *clValid*. Zbadamy skuteczność algorytmów *kmeans*, *diana*, *PAM*, *agnes* oraz *clara* dla klas $k = 2, ..., 6$
Najpierw zajmiemy się wskaźnikami wewnętrznymi. Tabela 19 zawiera informacje dotyczące algorytmów, dla których udało się osiągnąć najlepsze wyniki poszczególnych wskaźników.

Tabela 19: Algorytmy, dla których wskaźniki wewnętrzne są optymalne

	Score	Method	Clusters
Connectivity	3.8579365	agnes	2
Dunn	0.4933930	agnes	2
Silhouette	0.6342849	agnes	2





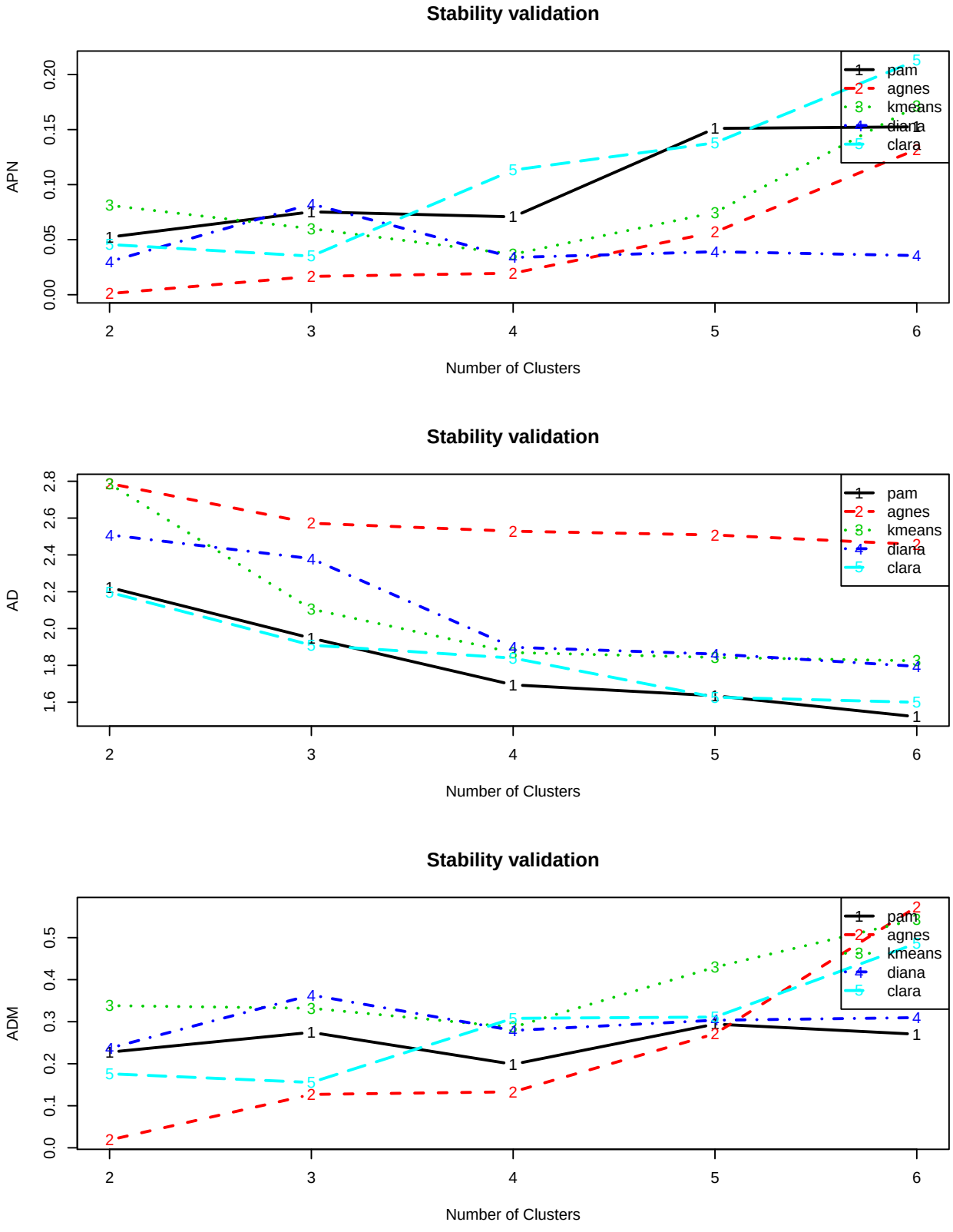


Rysunek 19: Porównanie skuteczności algorytmów w zależności od liczby klas

Tabela 20: Algorytmy, dla których wskaźniki stabilności są optymalne

	Score	Method	Clusters
APN	0.0010335	agnes	2
AD	1.5208284	pam	6
ADM	0.0185924	agnes	2
FOM	0.4916800	pam	6

Następnie zbadamy skuteczność algorytmów na wskaźnikach stabilności. Tabela 20 zawiera informacje na temat algorytmów, które osiągają najlepsze wyniki dla poszczególnych wskaźników stabilności.



Rysunek 20: Porównanie skuteczności algorytmów w zależności od liczby klas

2.4 Podsumowanie

- Dla zboru danych *Glass*, spośród zastosowanych funkcji, najlepsze rezultaty przyniósł algorytm hierarchiczny AGNES z metodą łączenia skupień *Warda*.
- Czas działania algorytmu AGNES był dłuższy niż w przypadku algorytmu PAM. Dla zbiorów danych zawierających wiele rekordów lub cech, wyznaczanie odpowiednich skupisk z wykorzystaniem algorytmów hierarchicznych mogłoby być czasochłonne.
- Otrzymujemy najbardziej zwarte i odseparowane od siebie klastry dla algorytmu agnes z metodą *complete*.
- Zbiór cech wykorzystany do wyznaczenia skupisk nie charakteryzuje się wysoką zdolnością dyskryminacyjną.