Raport 4

Eksploracja danych

Mikołaj Langner, Marcin Kostrzewa
nr albumów: 255716, 255749

2021-05-28

Spis treści

1 Wstęp

Niniejszy raport zawiera rozwiązania rozwiązania zadań z listy 4.

W zadaniu pierwszym zastosujemy zaawansowane metody klasyfikacji:

- bagging,
- boosting,
- random forest,
- metodę wektorów nośnych (SVM),

W zadaniu drugim badamy jakość

2 Zadanie 1

2.1 a)

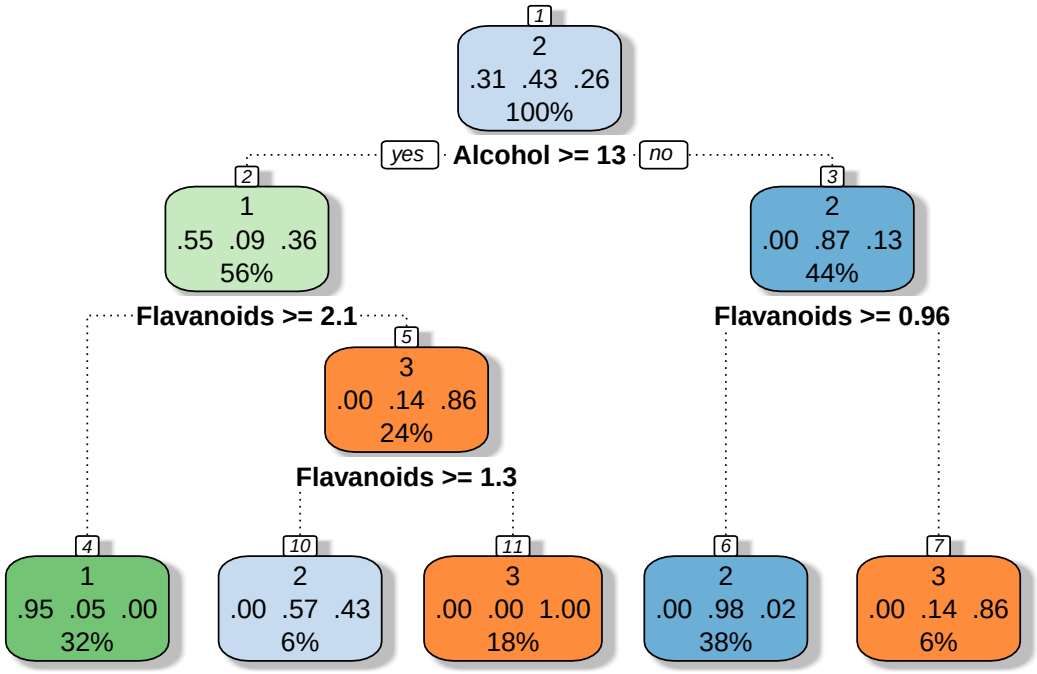
Naszym zadaniem będzie zbadanie tego jak wygląda

2.1.1 Pojedyncze drzewo decyzyjne

Przypomnijmy najpierw jak radziła sobie metoda drzewa klasyfikacyjnego.

```
tree.model <- rpart(Type ~ ., data = train.subset, cp=0)
```

Wyglądało ono następująco — rysunek (1).



Rysunek 1: Pojedyncze drzewo decyzyjne.

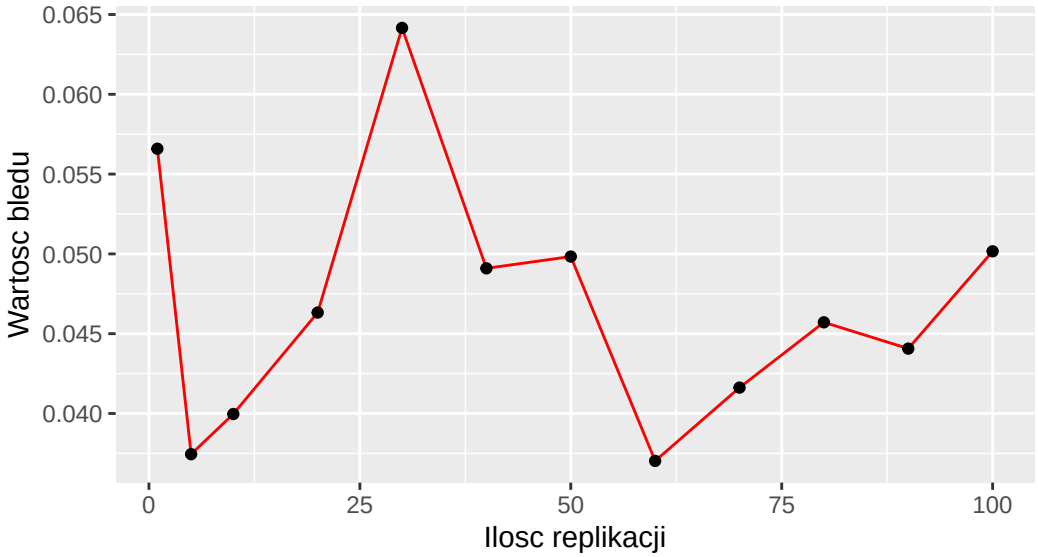
Błędy predykcji wyznaczone za pomocą metod: 5-krotnej walidacji krzyżowej, bootstrap oraz `.632+`, wyniosły kolejno 0.1179775, 0.1180151 oraz 0.1008182.

2.1.2 Bagging

Najpierw skorzystamy z algorytmu bagging. Znajdziemy optymalną wartość dla parametru `nbagg`.

```
B.vector <- c(1, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100)
bagging.error.rates <- sapply(B.vector, function(b)
```

```
  {errorest(Type~., data=train.data, model=bagging,
            nbagg=b, estimator="632plus",
            est.para=control.errorest(nboot = 20))$error})
choice <- B.vector[which.min(bagging.error.rates)]
```



Rysunek 2: Wplyw ilosci replikacji na blad klasyfikacji.

Jak widać, najlepiej zbudować model dla nbagg równego 60. Parametr złożoności `cp` przyjmujemy równy 0, tak jak w przypadku pojedynczego drzewa.

```
bagging.model <- bagging(Type~., data=train.data, nbagg=choice,
                         minsplit=1, cp=0)
bagging.train.pred <- predict(bagging.model, train.data)
bagging.test.pred <- predict(bagging.model, test.data)
```

Wyznaczymy dla tego modelu macierze pomyłek i wartości błędów klasyfkacji.

	1	2	3
1	36	0	0
2	0	51	0
3	0	0	31

(a) Zbior uczacy

	1	2	3
1	21	0	0
2	2	19	0
3	0	1	17

(b) Zbior testowy

Tabela 1: Macierze pomylek dla algorytmu bagging.

Błędy klasyfikacji to kolejno 0 i 0.05.

Wyznaczymy teraz dla tego modelu klasyfikacyjnego błędy predykcji podobnie jak dla drzewa decyzyjnego.

```
predictor  <- function(model, newdata)
{predict(model, newdata=newdata, type = "class")}

bagging.predictor <- function(formula, data)
{bagging(formula, data = data, nbagg = choice, cp = 0)}

bagging.error.cv <- errorest(Type~., wine,
                                    model=bagging.predictor,
                                    predict=predictor, estimator="cv",
                                    est.para=control.errorest(k = 5))
bagging.error.boot <- errorest(Type~., wine,
                                      model=bagging.predictor,
                                      predict=predictor, estimator="boot",
                                      est.para=control.errorest(nboot = 25))
bagging.error.632 <- errorest(Type~., wine,
                                      model=bagging.predictor,
                                      predict=predictor, estimator="632plus",
                                      est.para=control.errorest(nboot = 25))
```

Błędy wyniosły kolejno 0.0561798, 0.0561724 oraz 0.0298668.

2.1.3 Boosting

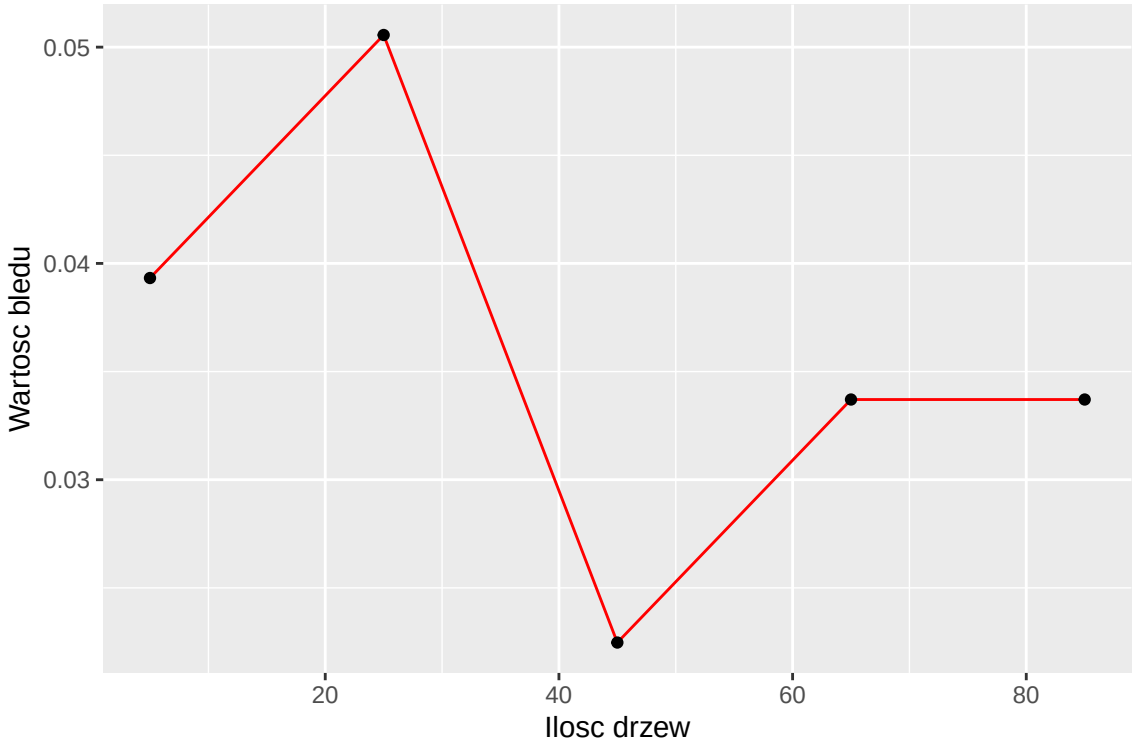
Wykorzystamy teraz algorytm boosting — skorzystamy z funkcji `boosting` z pakietu `adabag`.

Najpierw dobierzemy optymalnie wartość parametru `mfinal` — ilość wykorzystanych przez algorytm drzew.

```
mfinal.vector <- seq(5, 100, by=20)
predictor  <- function(model, newdata)
{ as.factor(predict(model, newdata=newdata, type = "class")$class)}
boosting.error.rates <- sapply(mfinal.vector, function(m)
  { errorest(Type~., wine, predict=predictor,
           model=boosting, mfinal=m,
           estimator="cv",
           est.para=control.errorest(k = 5))$error})
mfinal.choice <- mfinal.vector[which.min(boosting.error.rates)]
```

Wybieramy wartość mfinal równą 45.

```
boosting.model <- boosting(Type~., data = train.data, boos = TRUE,
                           mfinal = mfinal.choice)
boosting.test.pred <- predict(boosting.model, test.data)
boosting.train.pred <- predict(boosting.model, train.data)
```



Rysunek 3: Zaleznosc bledu od ilosci drzew.

```
test.confusion <- boosting.test.pred$confusion
train.confusion <- boosting.train.pred$confusion

print(xtable(train.confusion), file="boost1.tex", floating=FALSE)
print(xtable(test.confusion), file="boost2.tex", floating=FALSE)
```

	1	2	3
1	36	0	0
2	0	51	0
3	0	0	31

(a) Zbior uczacy

	1	2	3
1	22	0	0
2	1	19	0
3	0	1	17

(b) Zbior testowy

Tabela 2: Macierze pomylek dla algorytmu boosting.

Błędy klasyfikacji to kolejno 0 i 0.0333333.

Wyznaczymy też teraz błędy predykcji (tym razem z racji złożoności obliczeniowej i długiego czasu wykonania tylko dla metody 5-krotnej walidacji krzyżowej).

```
boosting.predictor <- function(formula, data)
{ boosting(formula, data = data, mfinal=mfinal.choice, boos = TRUE)}
```

```
boosting.error.cv <- errorest(Type~., wine,
                                    model=boosting.predictor,
                                    predict=predictor, estimator="cv",
                                    est.para=control.errorest(k = 5))
```

Błąd wyniosł 0.0224719.

2.1.4 Random Forest

Teraz wykorzystamy algorytm random forest.

Postaramy się odpowiednio dobrać parametry `ntree` (ilość drzew) i `mtry` (ilość losowo wybieranych cech).

```
ntree.vector <- seq(10, 500, by=20)
ntree.error.rates <- sapply(ntree.vector, function(b)
  {errorest(Type~., data=train.data, model=randomForest,
            ntree=b, estimator="632plus",
            est.para=control.errorest(nboot = 20))$error})
ntree.choice <- ntree.vector[which.min(ntree.error.rates)]

mtry.vector <- seq(1, sqrt(ncol(wine))+1, by=1)
mtry.error.rates <- sapply(mtry.vector, function(m)
  {errorest(Type~., data=train.data, model=randomForest, ntree = ntree.choice,
            mtry=m, estimator="632plus",
            est.para=control.errorest(nboot = 20))$error})
mtry.choice <- mtry.vector[which.min(mtry.error.rates)]
```

Podobnie jak wcześniej wyznaczamy za pomocą modelu etykietki klas i wyznaczamy macierze pomyłek i błędy klasyfikacji.

```
rf.model <- randomForest(Type~., data = train.data, ntree=ntree.choice,
                         mtry=mtry.choice, importance=TRUE)
rf.test.pred <- predict(rf.model, test.data)
rf.train.pred <- predict(rf.model, train.data)
```

	1	2	3
1	36	0	0
2	0	51	0
3	0	0	31

(a) Zbior uczacy

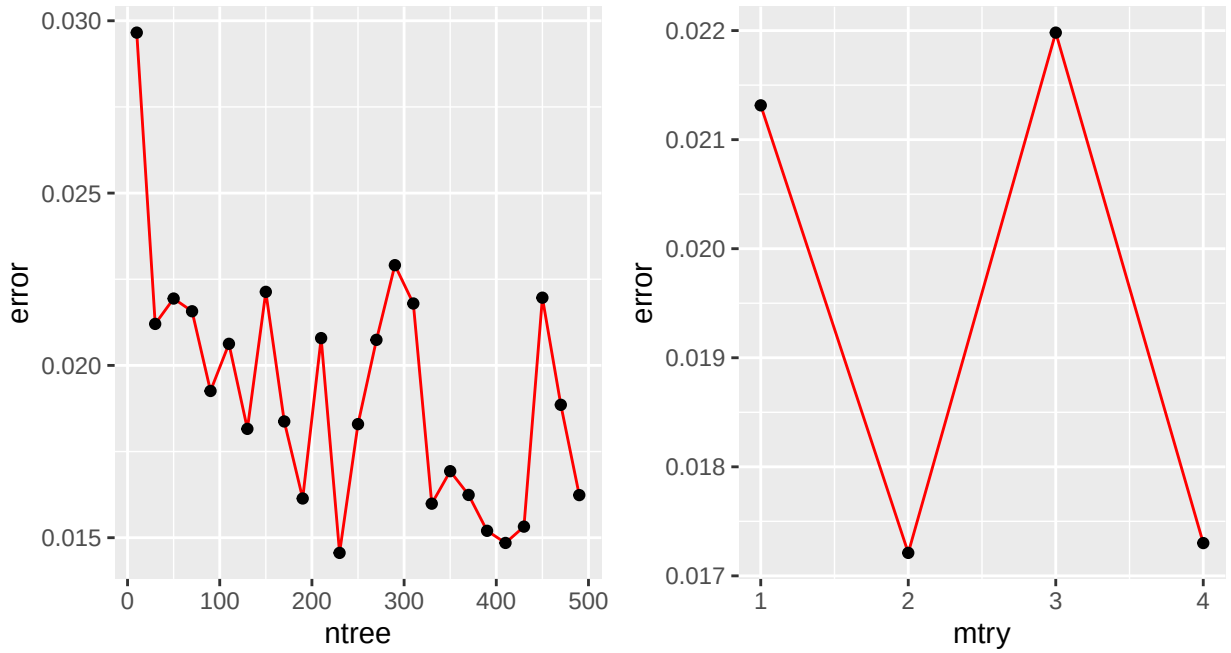
	1	2	3
1	23	0	0
2	0	19	0
3	0	1	17

(b) Zbior testowy

Tabela 3: Macierze pomylek dla algorytmu randomForest.

Błędy klasyfikacji to kolejno 0 i 0.0166667.

Tak jak dla wcześniejszych algorytmów, wyznaczymy teraz błędy predykcji.



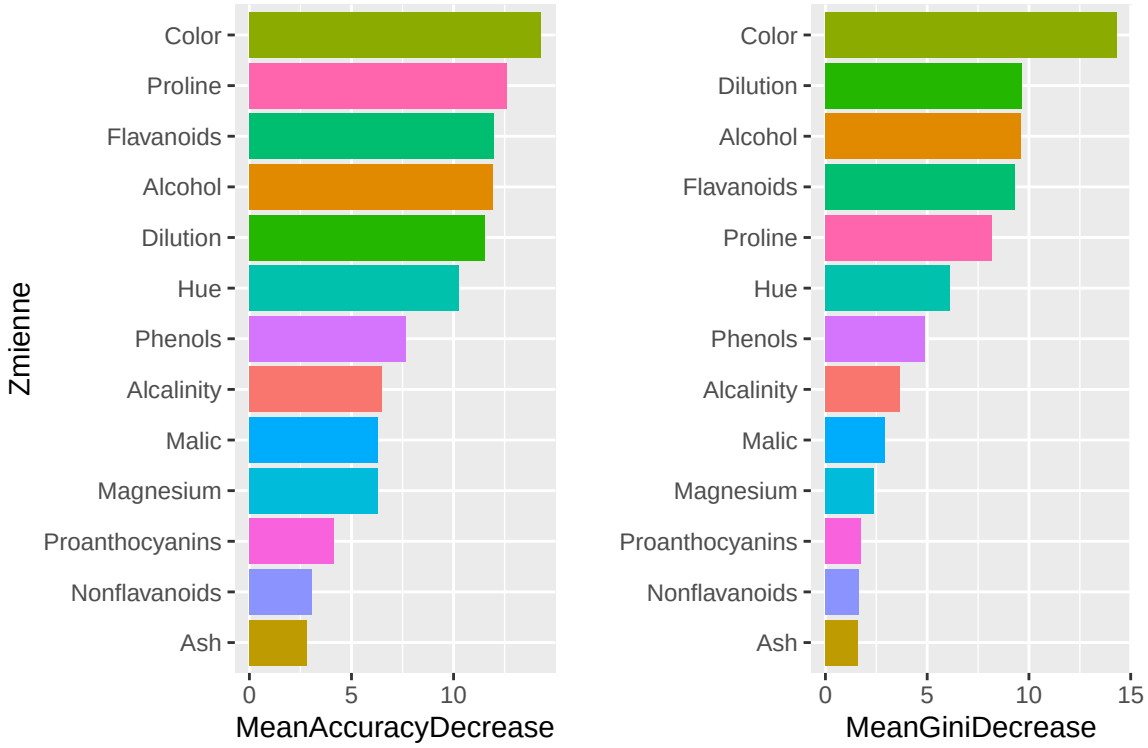
Rysunek 4: Wykresy zaleznosci bledu klasyfikacji od parametrow mtry i ntree.

```
predictor  <- function(model, newdata)
{ predict(model, newdata=newdata, type = "class") }

rf.predictor <- function(formula, data)
{ randomForest(formula, data = data, ntree = ntree.choice, mtry = mtry.choice)}

rf.error.cv <- errorest(Type~., wine, model=rf.predictor,
                                    predict=predictor, estimator="cv",
                                    est.para=control.errorest(k = 5))
rf.error.boot <- errorest(Type~., wine, model=rf.predictor,
                                      predict=predictor, estimator="boot",
                                      est.para=control.errorest(nboot = 25))
rf.error.632 <- errorest(Type~., wine, model=rf.predictor,
                                      predict=predictor, estimator="632plus",
                                      est.para=control.errorest(nboot = 25))
```

Wyniosły one kolejno 0.0168539, 0.0183539 oraz 0.0106015.

Wykorzystamy teraz algorytm random forest do wyznaczenia rankingu cech (*variable importance*).

Przypomnijmy, że na ostatniej liście za zmienne istotne, takie, które dobrze dywersyfikowały klasy ze zbioru `wine`, były `Alcohol` i `Flavanoids`. Tak jak widzimy to na rysunku (5) dokonaliśmy wtedy całkiem dobrej decyzji, ponieważ zmienne te są w czołówce najważniejszych zmiennych. Wykresy wskazują, że najważniejsze są zmienne `Color` i `Proline`, które w trakcie



Rysunek 5: Wykres waznosci zmiennych.

dokonywania wyboru, odrzuciliśmy.

2.1.5 Wnioski

uwagii, tabelka, wniosek, że klasyfikatory wzmocnione radzą sobie lepiej

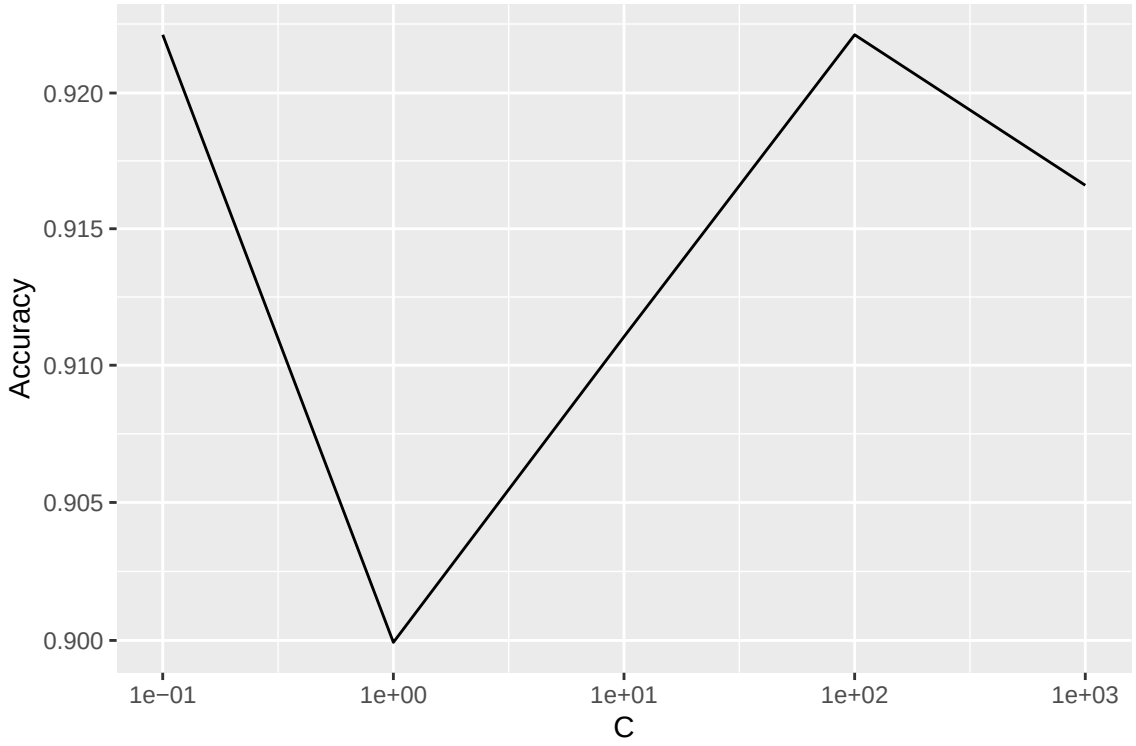
2.2 b)

```
wine <- wine %>% select(c(Type, Alcohol, Flavanoids))
```

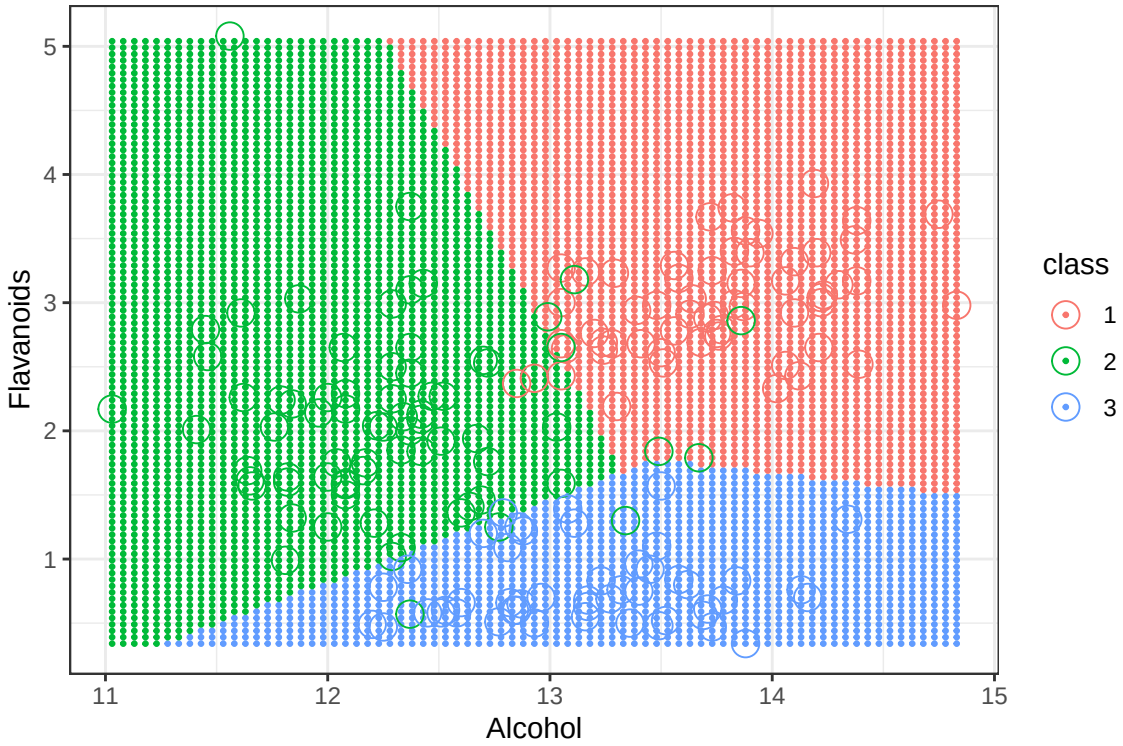
```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

linear	polynomial	radial
0.928	0.938	0.933

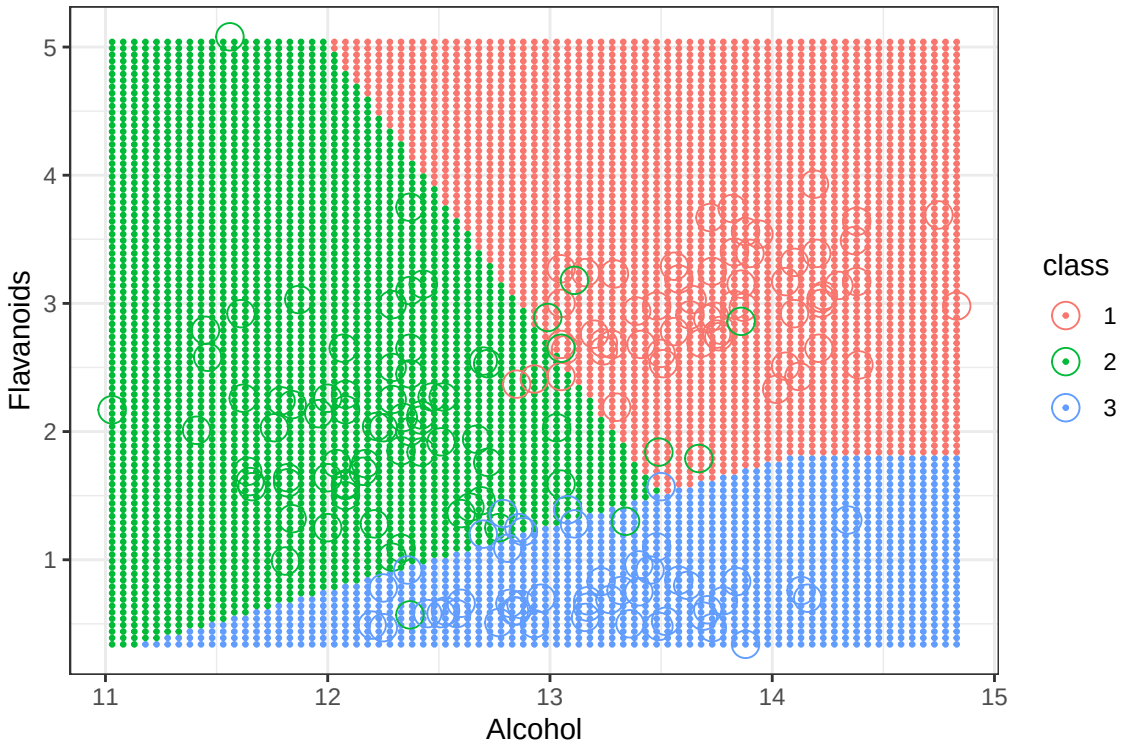
Tabela 4: Porównanie klasyfikatorów dla różnych jąder



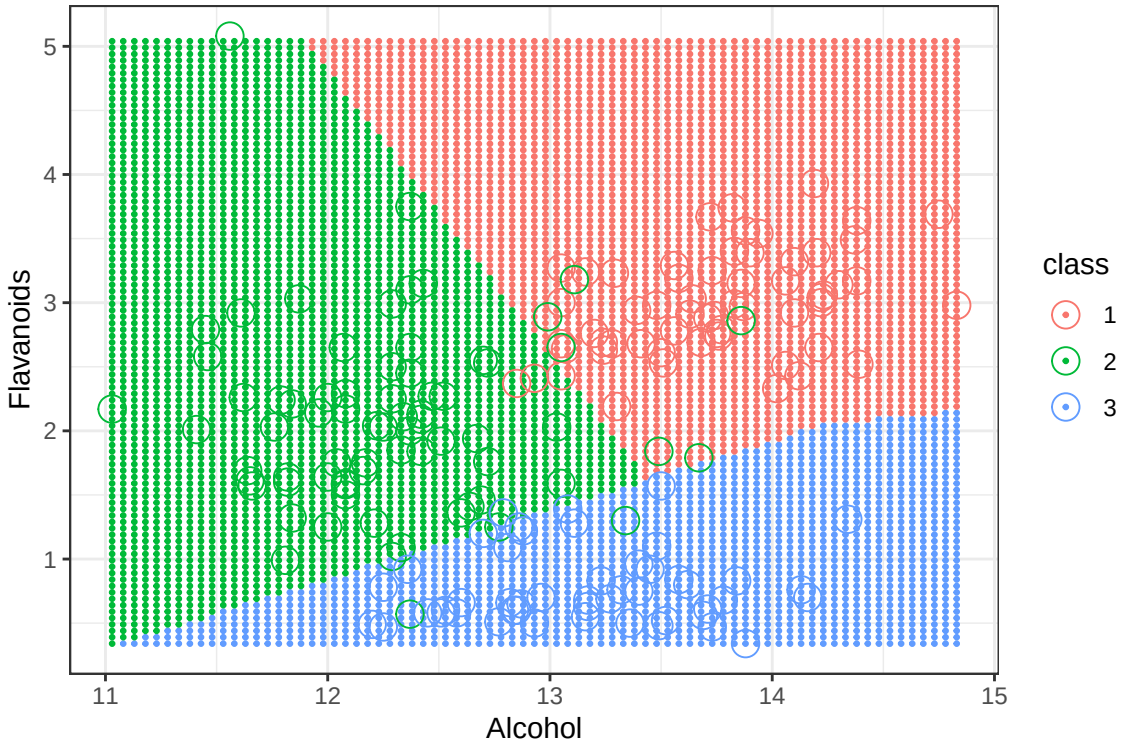
Rysunek 6: Dokładność klasyfikatora od parametru kosztu



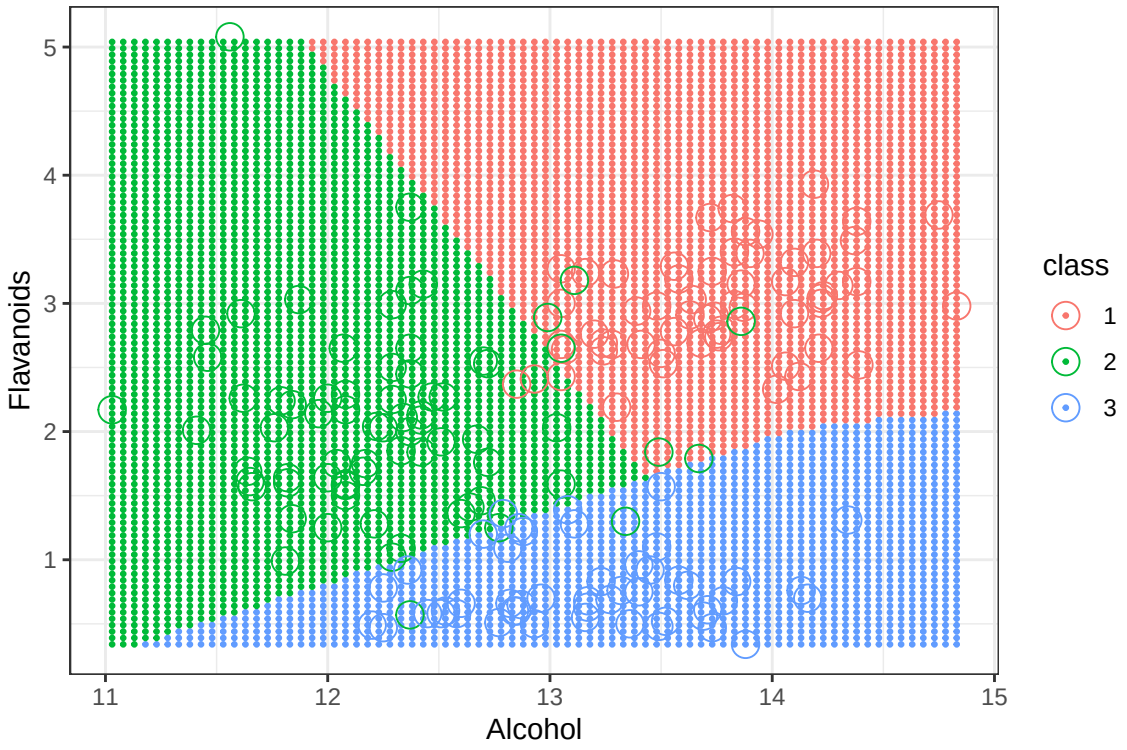
Rysunek 7: Obszary decyzyjne dla $C = 0.1$



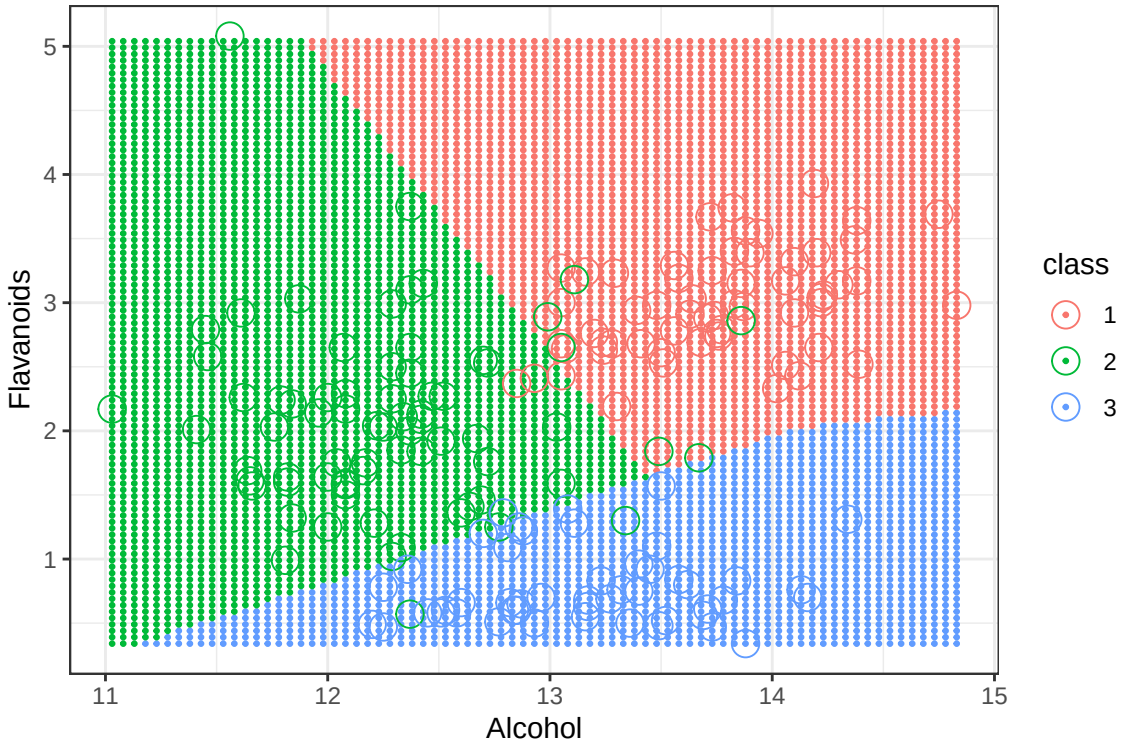
Rysunek 8: Obszary decyzyjne dla $C = 1$



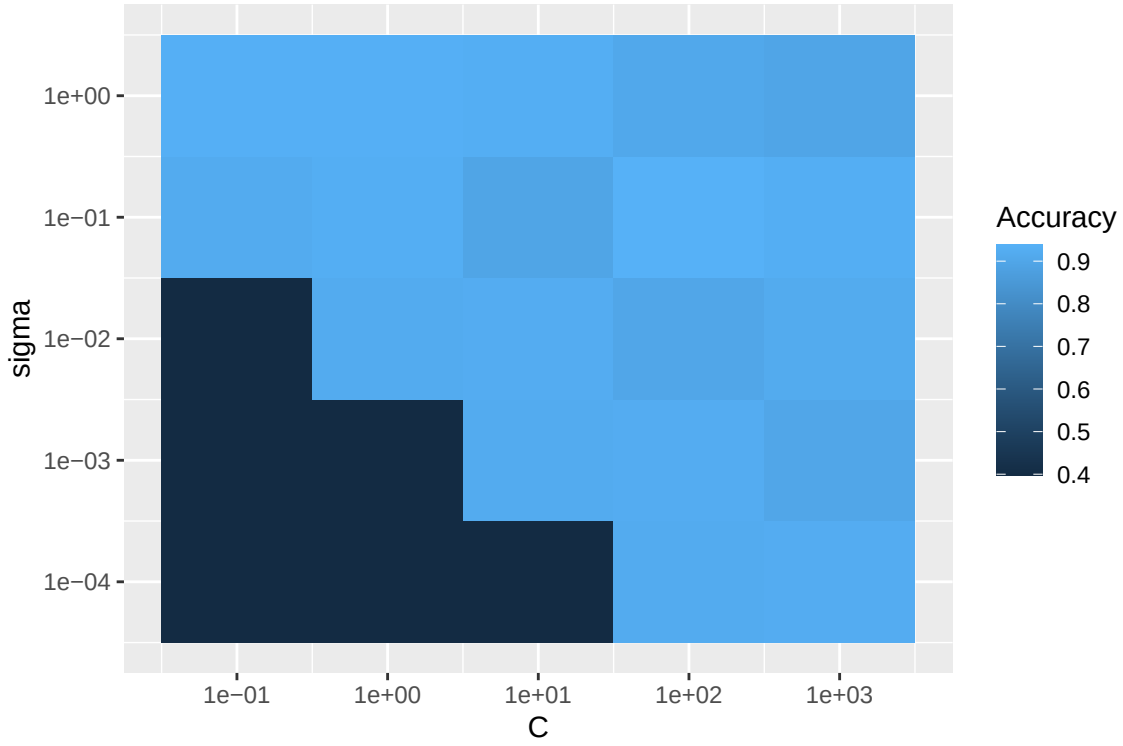
Rysunek 9: Obszary decyzyjne dla $C = 10$



Rysunek 10: Obszary decyzyjne dla $C = 100$



Rysunek 11: Obszary decyzyjne dla $C = 1000$



Rysunek 12: Mapa ciepła dokładności klasyfikatora

sigma	C
0.10	100.00

Tabela 5: Parametry dla najlepszego klasyfikatora

3 Zadanie 2

W tym zadaniu zastosujemy algorytmy analizy skupień do wyznaczenia klastrów dla zbioru `wine`, ocenimy ich skuteczność i porównamy je ze sobą. Sięgniemy po dwa algorytmy: PAM i AGNES.

3.1 Wizualizacja wyników grupowania ($K = 3$)

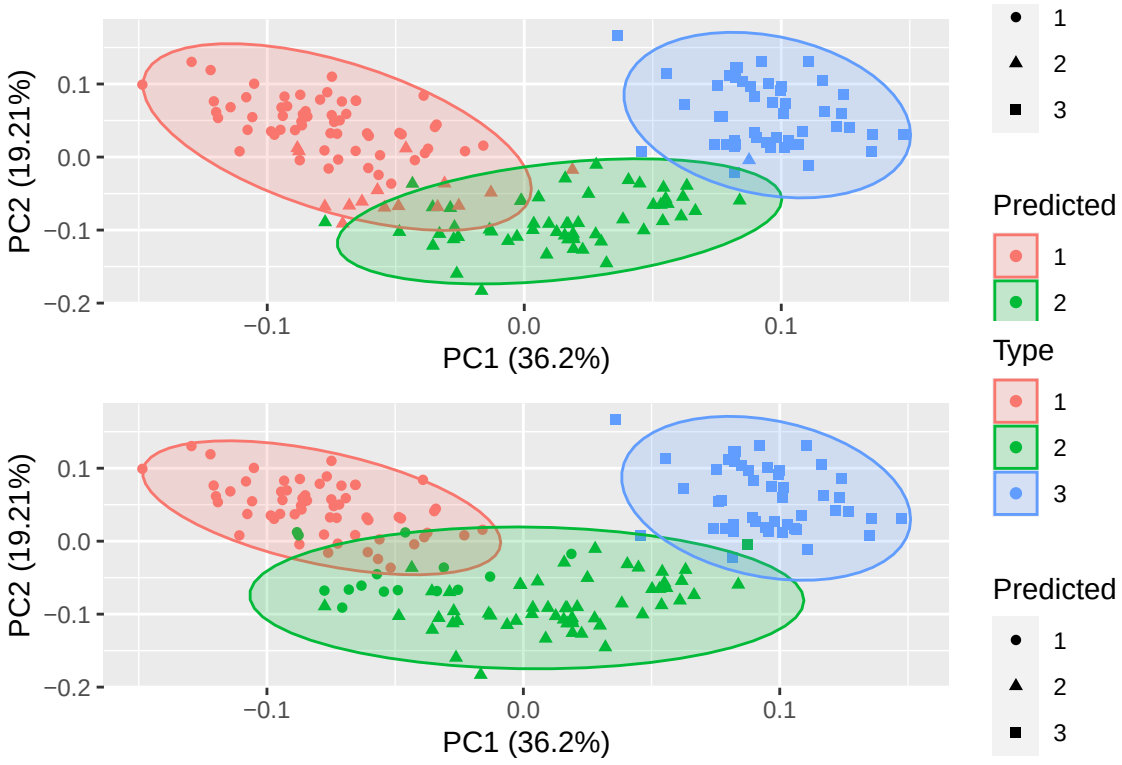
Przyjrzyjmy się najpierw jakie wyniki daje nam zastosowanie algorytmu PAM.

Zobaczmy teraz, jak poradził sobie algorytm AGNES z single-linkage.

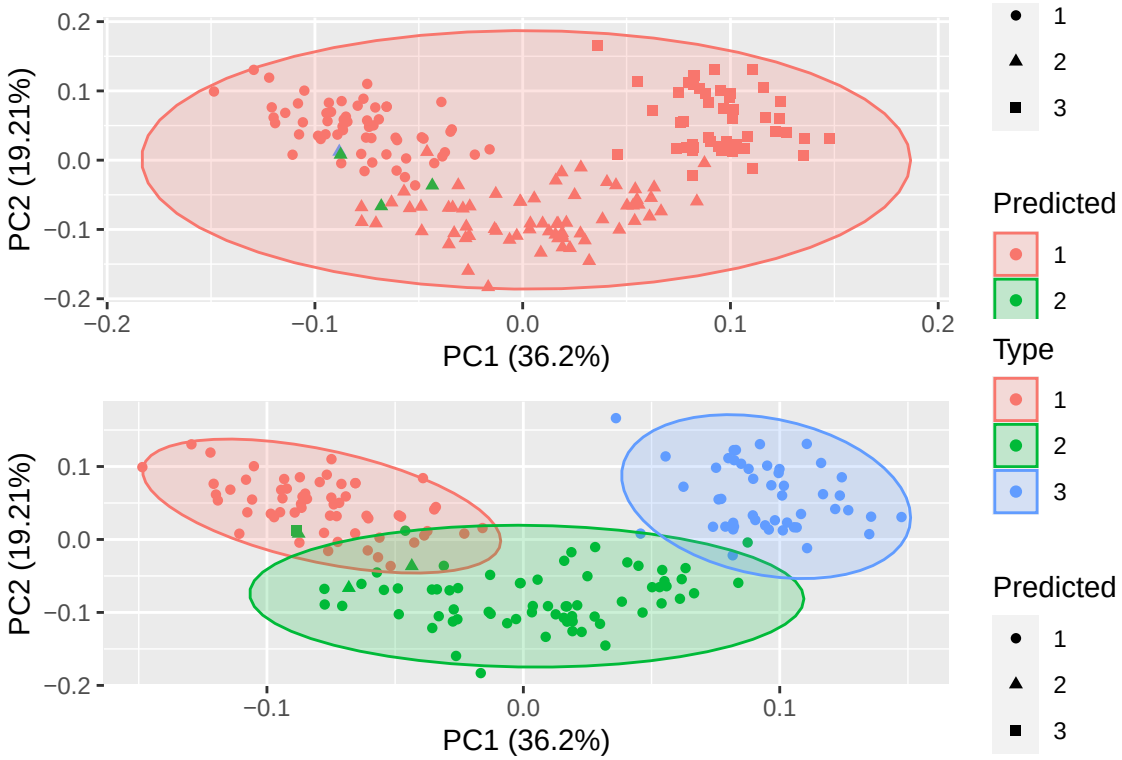
Poniżej wyniki dla algorytmu AGNES z complete-linkage.

3.2 Ocena jakości grupowania

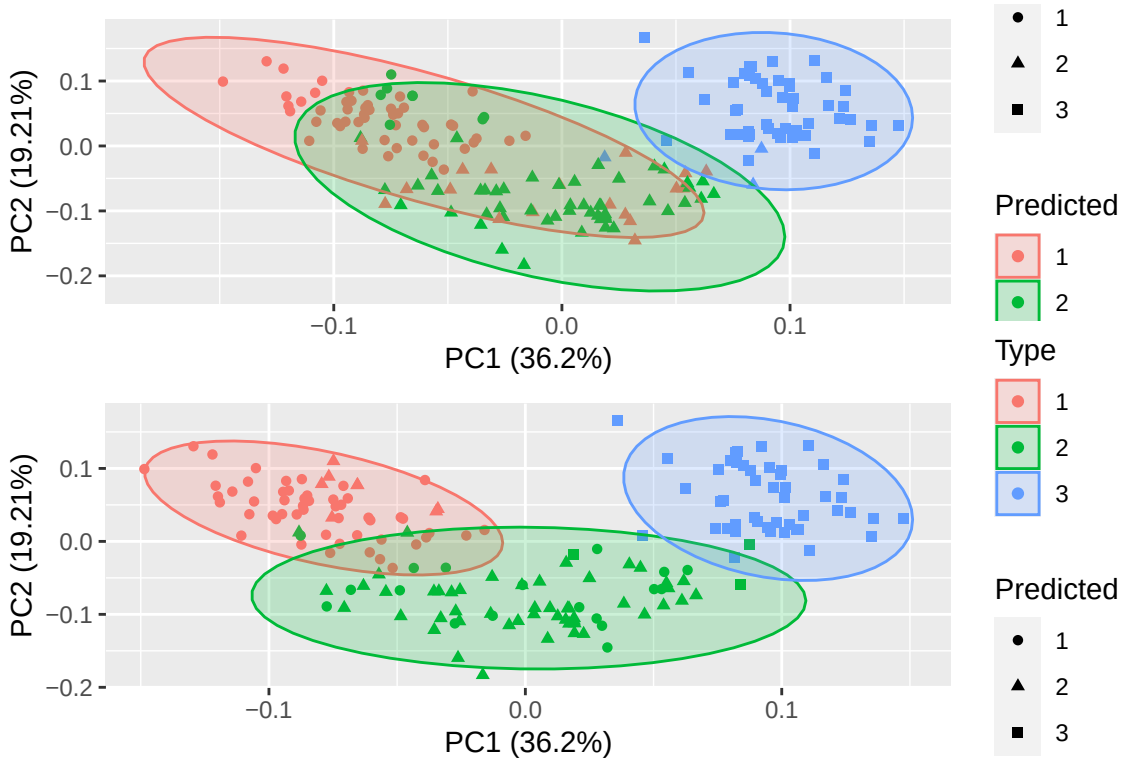
W tej części zadania, porównamy ze sobą algorytmy, jakość uzyskanego dzięk nim grupowania w zależności od przyjętej ilości skupień. Wykorzystamy wskźniki wewnętrzne, jak i zewnętrzne.



Rysunek 13: Skupienia dla metody PAM



Rysunek 14: Skupienia dla metody AGNES z single-linkage



Rysunek 15: Skupienia dla metody AGNES z complete-linkage

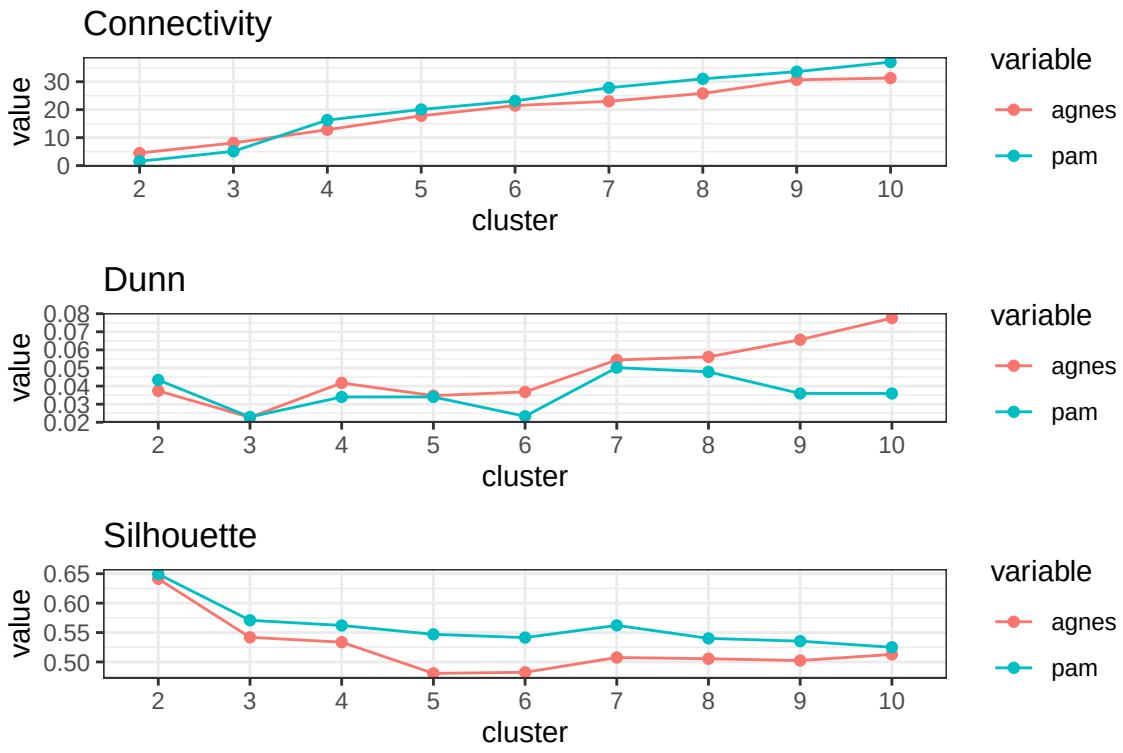
3.2.1 Wkaźniki wewnętrzne

```
##
## Clustering Methods:
##  agnes pam
##
## Cluster sizes:
##  2 3 4 5 6 7 8 9 10
##
## Validation Measures:
##                            2       3       4       5       6       7       8       9
##
## agnes Connectivity   4.4925  8.0972 12.8210 17.7913 21.4591 22.9877 25.8044 30.6730 3
##       Dunn           0.0374  0.0227  0.0417  0.0347  0.0368  0.0544  0.0561  0.0656
##       Silhouette     0.6413  0.5419  0.5336  0.4806  0.4824  0.5075  0.5055  0.5024
## pam   Connectivity   1.5286  5.1048 16.2798 20.0643 23.1155 27.8393 31.0163 33.5841 3
##       Dunn           0.0434  0.0229  0.0340  0.0340  0.0233  0.0502  0.0478  0.0359
##       Silhouette     0.6494  0.5708  0.5620  0.5469  0.5414  0.5622  0.5401  0.5353
##
## Optimal Scores:
##
##              Score  Method Clusters
```

```
## Connectivity 1.5286 pam    2
## Dunn         0.0776 agnes  10
## Silhouette   0.6494 pam    2
```

```
##                    Score Method Clusters
## Connectivity 1.52857143    pam        2
## Dunn         0.07755693  agnes       10
## Silhouette   0.64936476    pam        2
```

```
## Using cluster as id variables
## Using cluster as id variables
## Using cluster as id variables
```

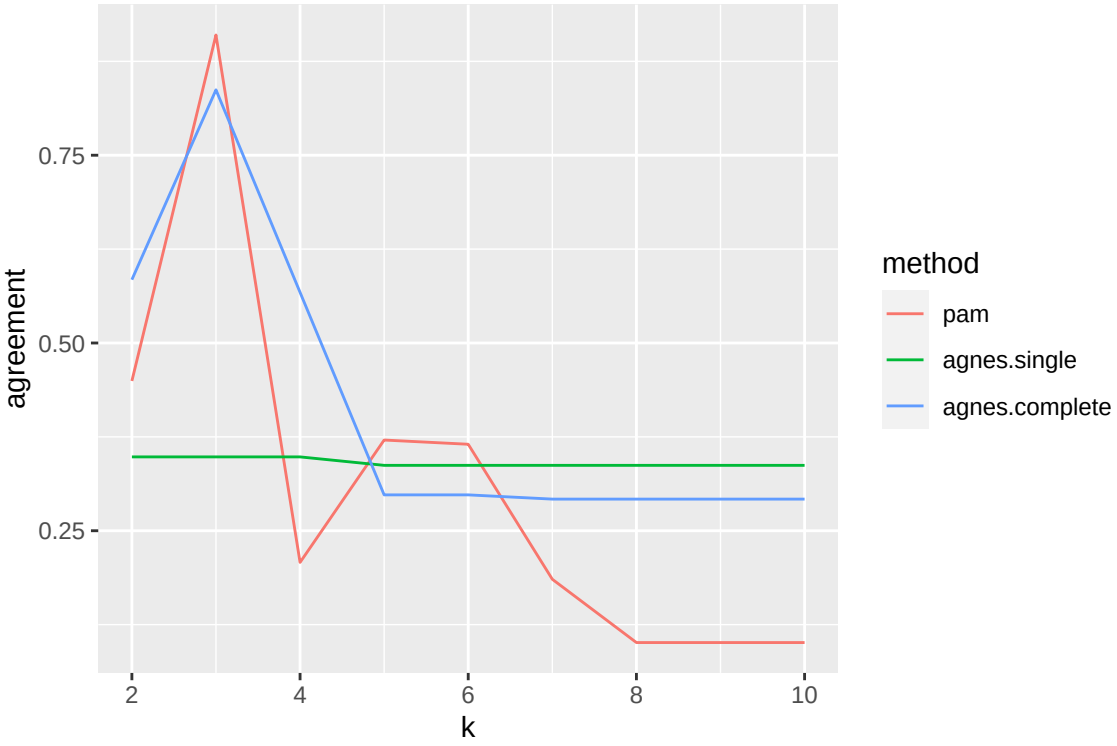


Rysunek 16: Wskaźniki wewnętrzne dla PAM i AGNES z complete-linkage

3.2.2 Wskaźniki zewnętrzne

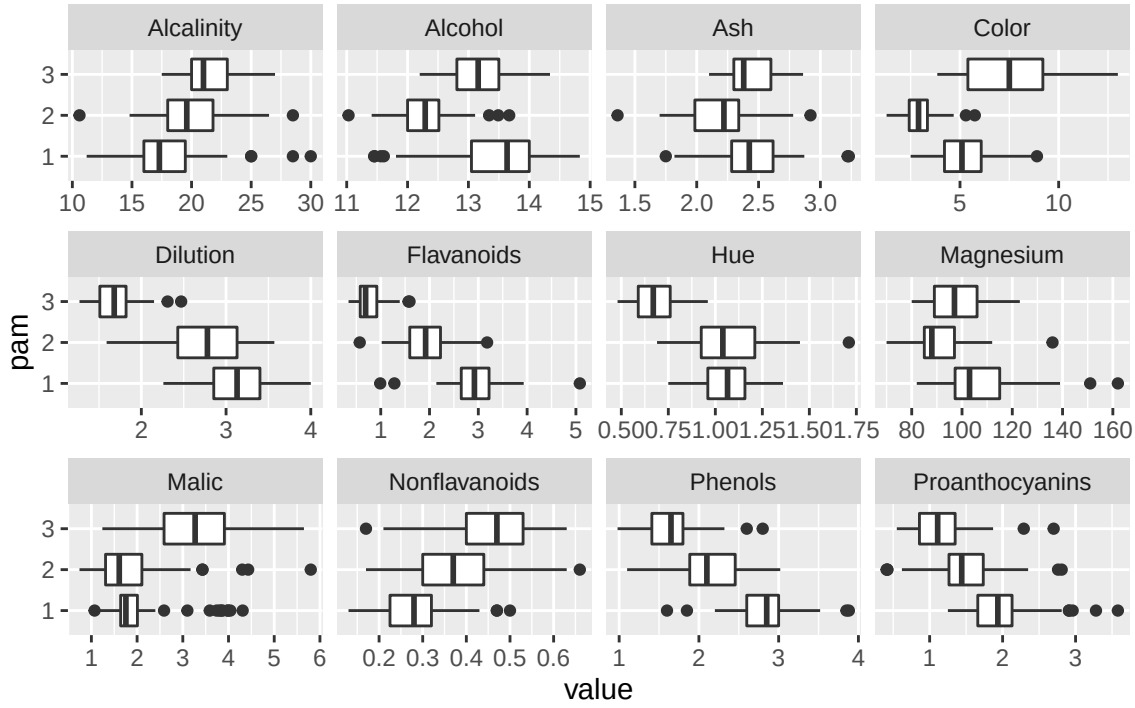
```
## Cases in matched pairs: 80.9 %
```

```
## 1 2 3
## 1 2 3
```

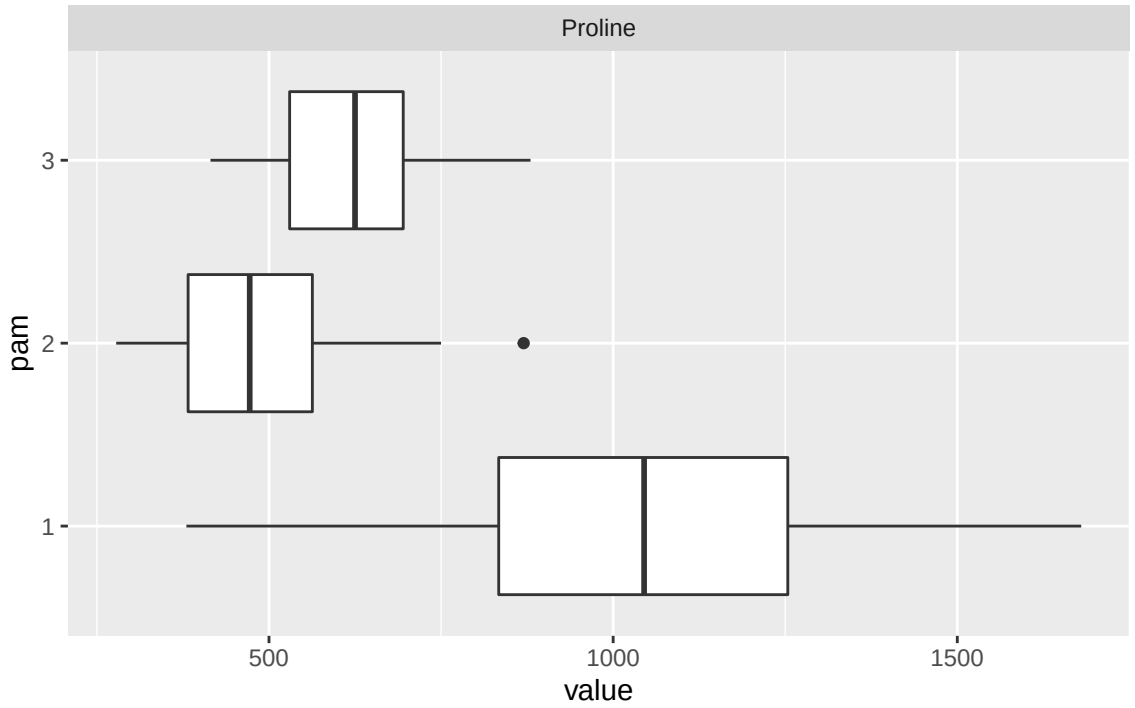


Rysunek 17: Porównanie wskaźników zewnętrznych

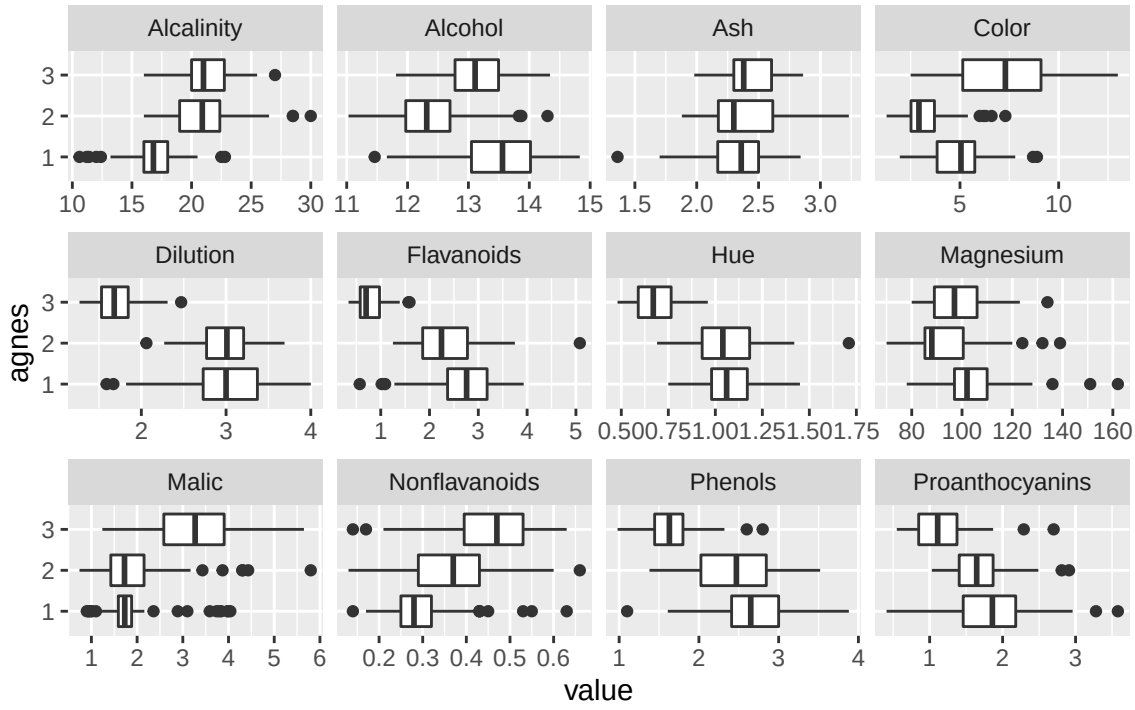
3.2.3 Ocena otrzymanych rezultatów



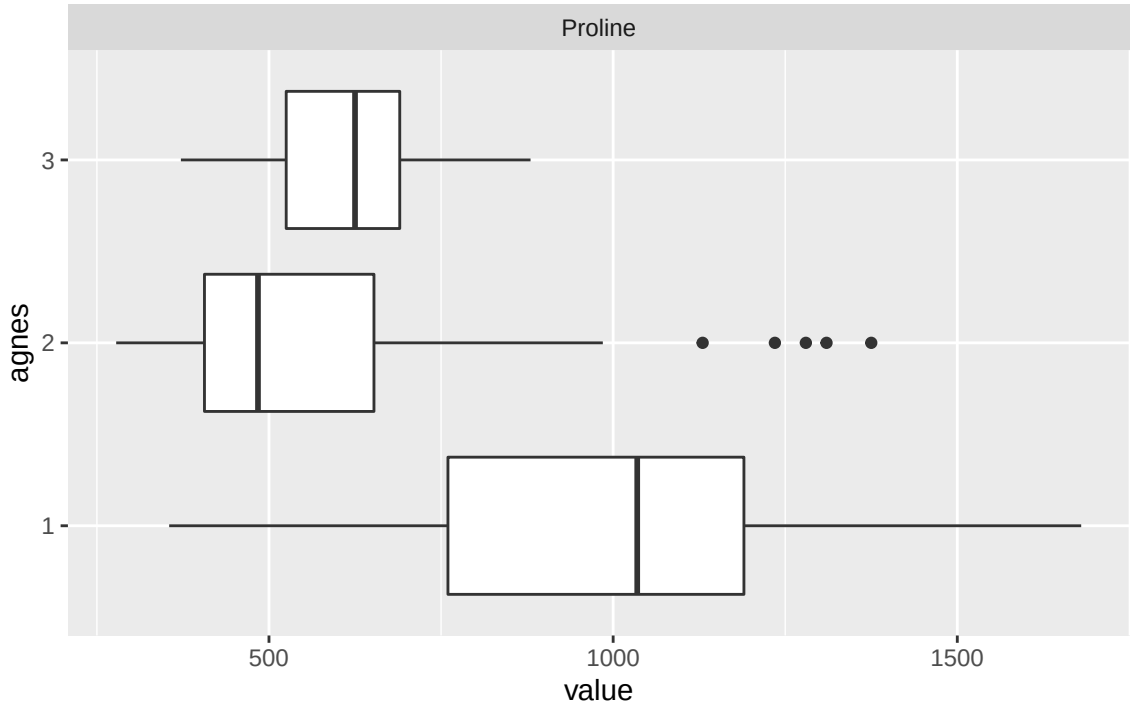
Page 1



Page 2


Alcalinity
Alcohol
Ash
Color
Dilution
Flavanoids
Hue
Magnesium
Malic
Nonflavanoids
Phenols
Proanthocyanins
agnes
value

Page 1


Proline
agnes
value

Page 2

	1	2	3
Alcohol	0.59	-0.92	0.39
Malic	-0.47	-0.54	0.81
Ash	0.16	-0.90	0.05
Alcalinity	0.30	-0.15	0.60
Magnesium	0.02	-1.38	-0.54
Phenols	0.65	-1.03	-0.58
Flavanoids	0.95	0.00	-1.27
Nonflavanoids	-0.82	0.07	0.71
Proanthocyanins	0.47	0.07	-0.60
Color	0.02	-0.72	1.45
Hue	0.36	0.19	-1.78
Dilution	1.21	0.79	-1.40
Proline	0.55	-0.75	-0.31

Tabela 6: Medoidy dla metody PAM przy K=3