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

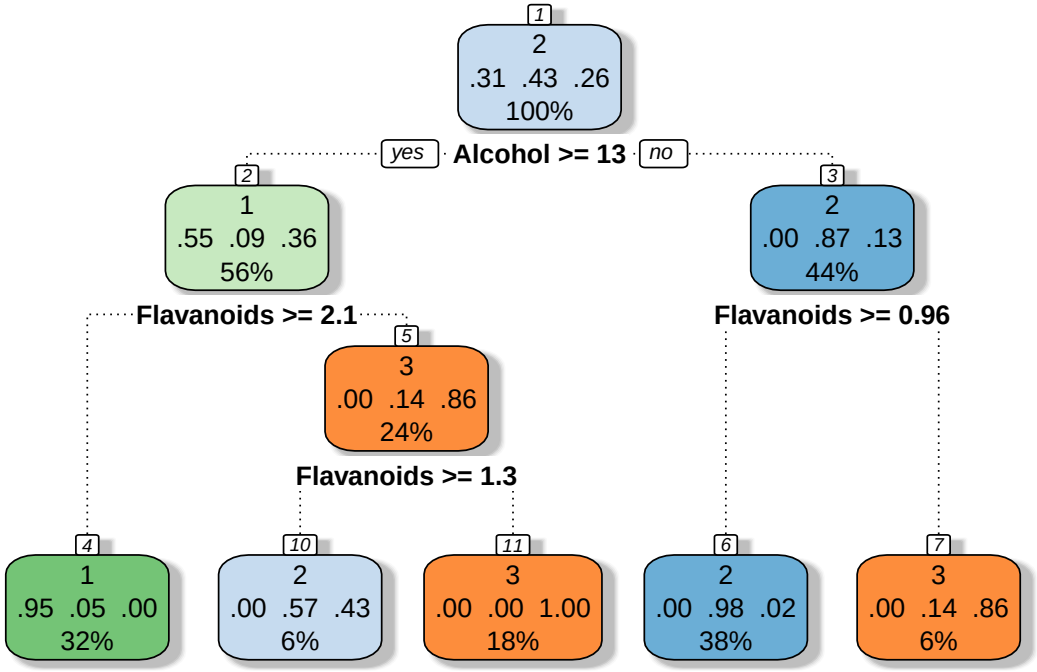
2.1 a)

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 (??).



Rysunek 1: Pojedyncze drzewo decyzyjne.

Przypomnimy także jak wyglądały błędy klasyfikacji dla drzewa.

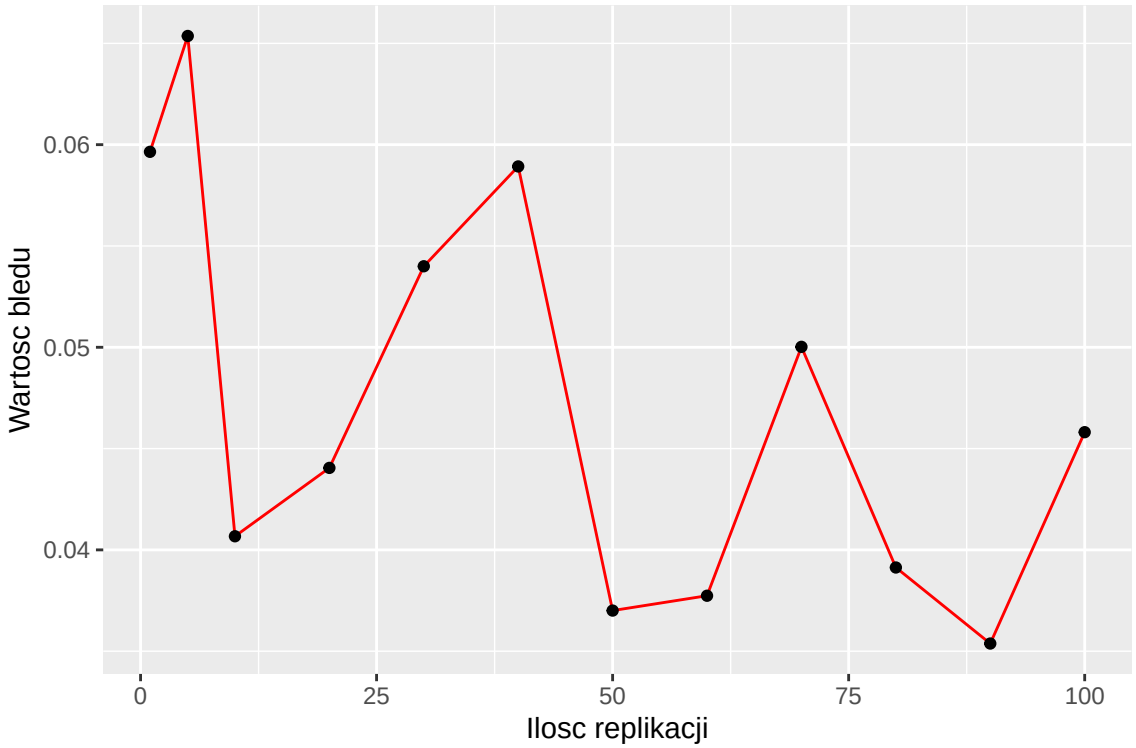
```
predictor  <- function(model, newdata)
{ predict(model, newdata=newdata, type = "class") }

decision.tree.predictor <- function(formula, data)
{ rpart(formula, data = data, cp = 0)}

decision.tree.error.cv <- errorest(Type~., wine[, c(1, 2, 8)],
                                   model=decision.tree.predictor,
                                   predict=predictor, estimator="cv",
                                   est.para=control.errorest(k = 5))
```

Wyniósł on 0.1179775.

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 90.

```
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.

Błędy klasyfikacji to kolejno 0 i 0.05.

	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.

Wyznaczymy teraz dla tego modelu klasyfikacyjnego błąd predykcji — skorzystamy z 5-krotnej walidacji krzyżowej, metody bootstrap oraz .632+.

```
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, 0.0584677 oraz 0.0402642.

2.1.3 Boosting

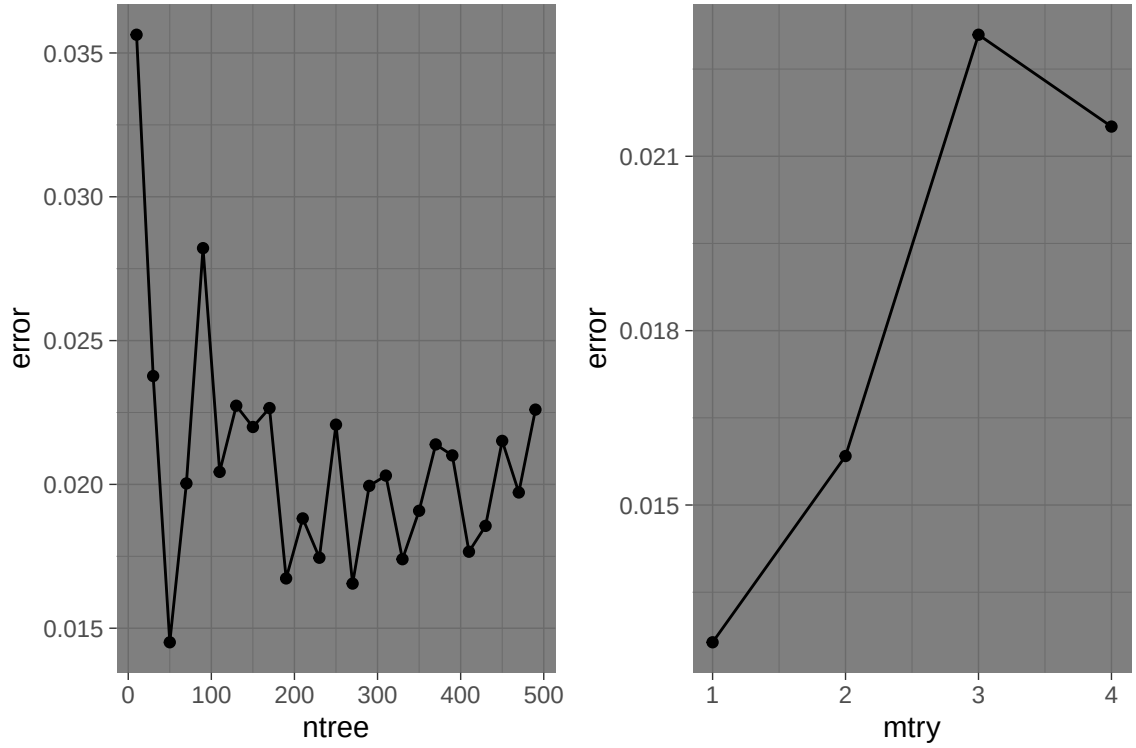
2.1.4 Random Forest

Teraz wykorzystamy algorytm random forrest.

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)]
```



Rysunek 3: Wykresy zaleznosci bledu klasyfikacji od parametrow mtry i ntree.

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)
```

Błędy klasyfikacji to kolejno 0 i 0.

Tak jak wcześniej wyznaczymy dla tego modelu błędy predykcji.

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

	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	20	0
3	0	0	17

(b) Zbior testowy

Tabela 2: Macierze pomylek dla algorytmu randomForest.

```
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))
```

Błędy wyniosły kolejno 0.005618, 0.0227064 oraz 0.0125766.

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

Widzimy, że ...

2.1.5 Wnioski

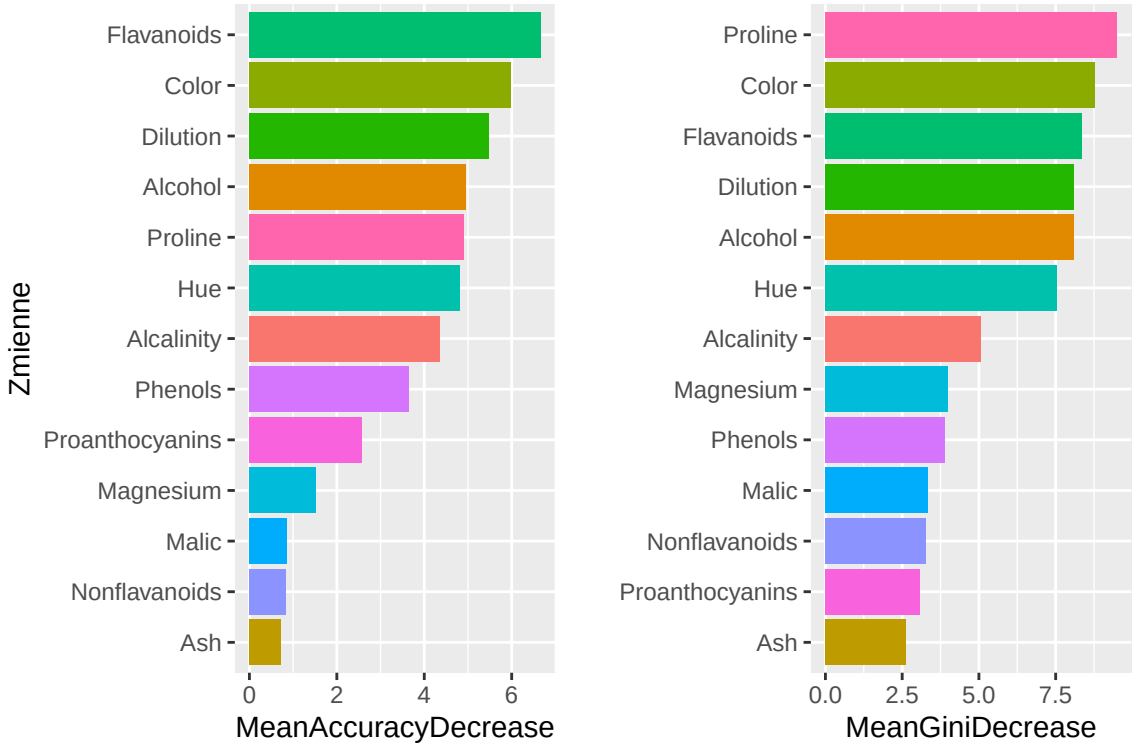
2.2 b)

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

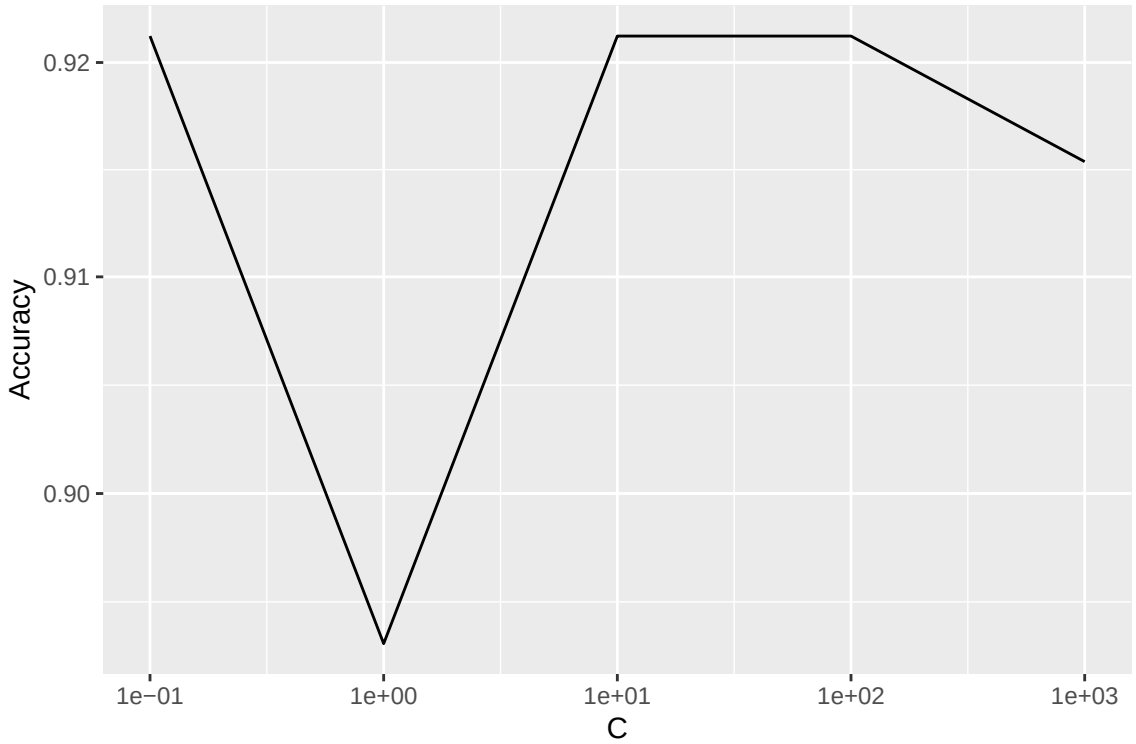
```
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
##  Setting default kernel parameters
```

% latex table generated in R 4.1.0 by xtable 1.8-4 package % Sat Jun 19 15:17:43 2021

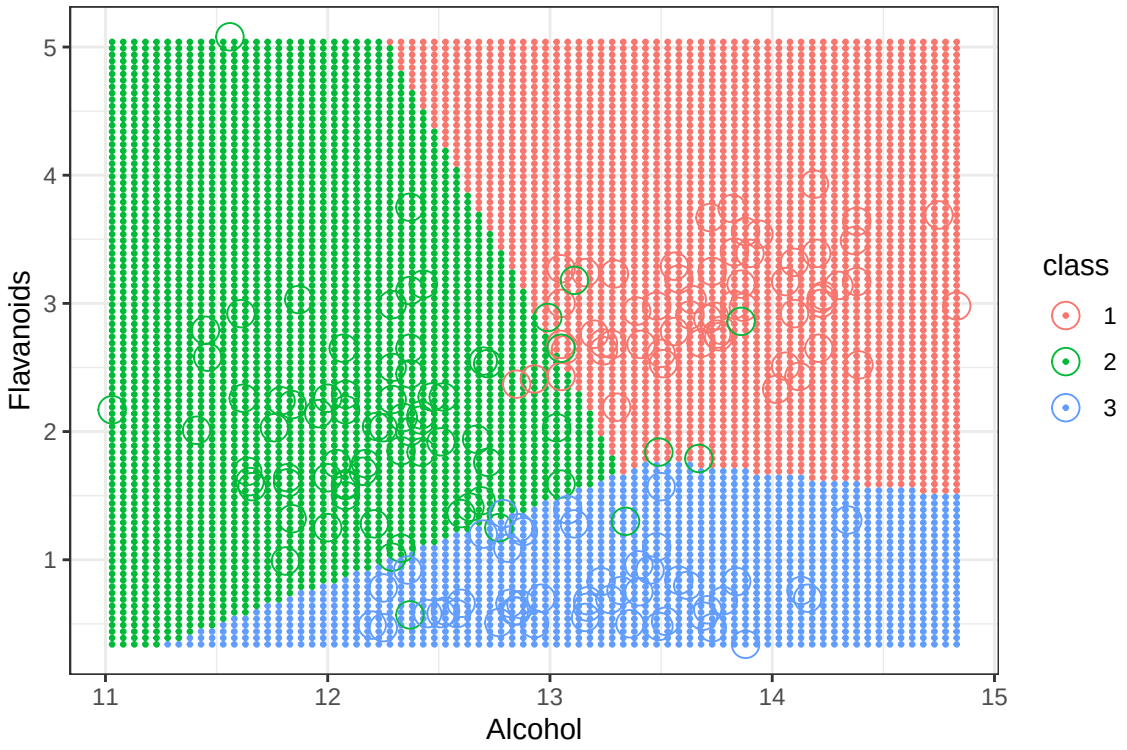
% latex table generated in R 4.1.0 by xtable 1.8-4 package % Sat Jun 19 15:17:46 2021



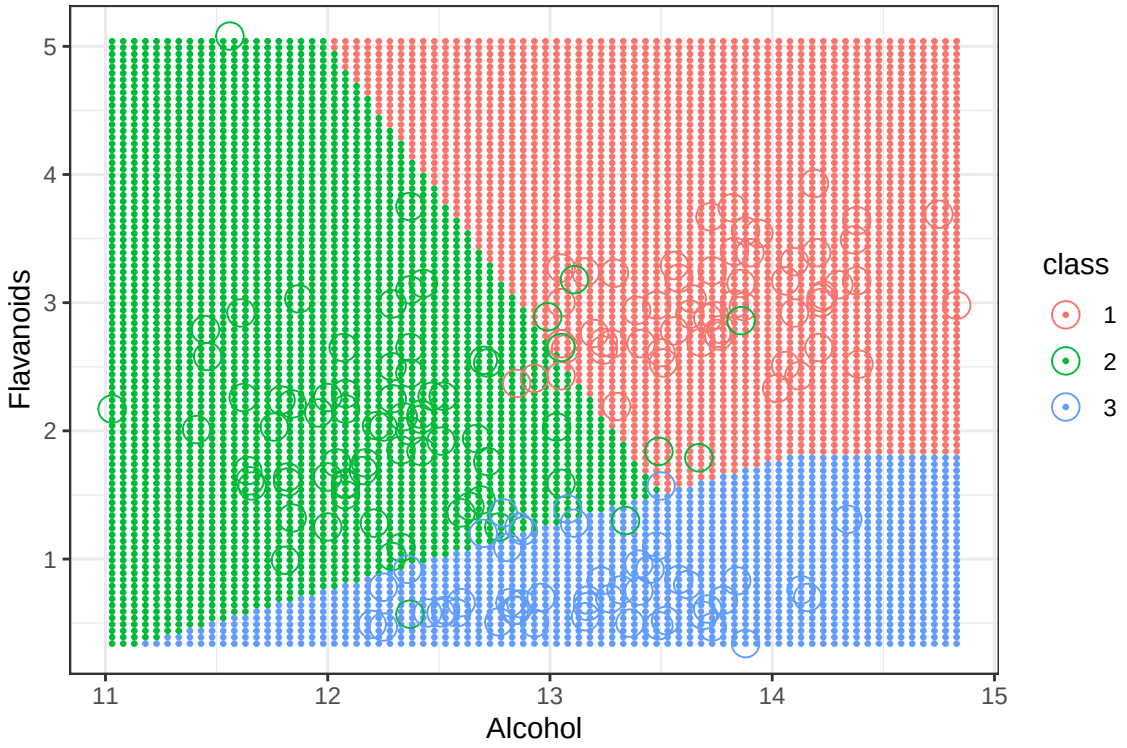
Rysunek 4: Wykres waznosci zmiennych.



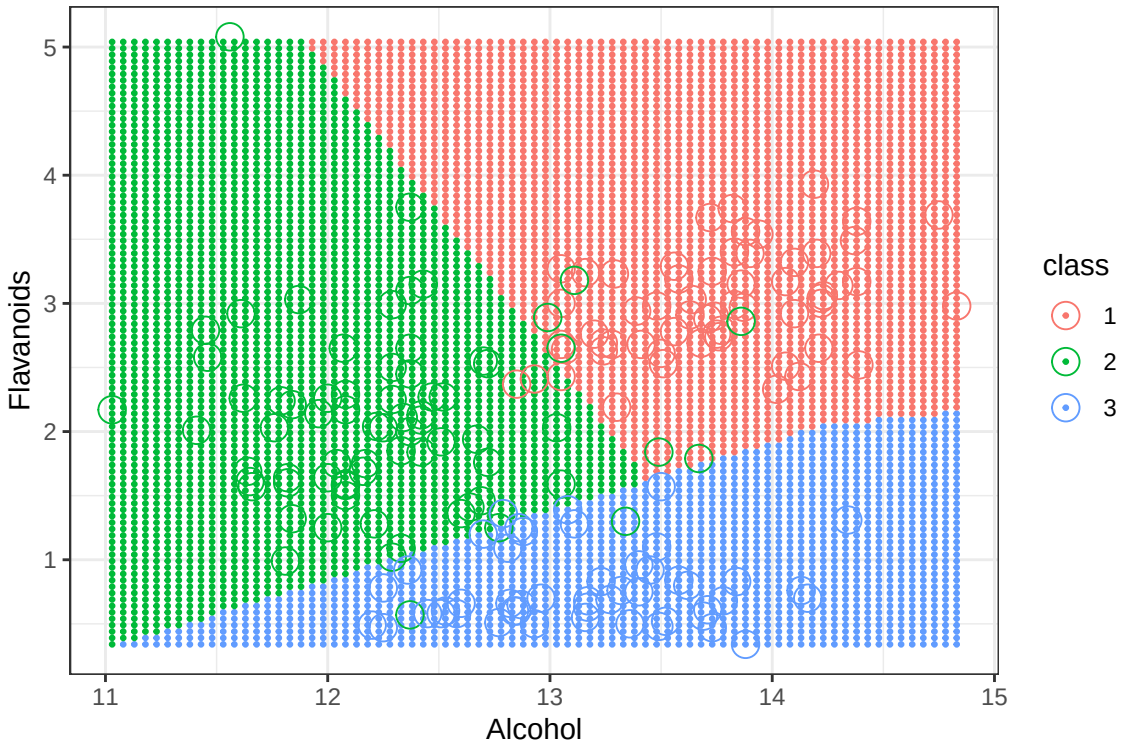
Rysunek 5: Dokładność klasyfikatora od parametru kosztu



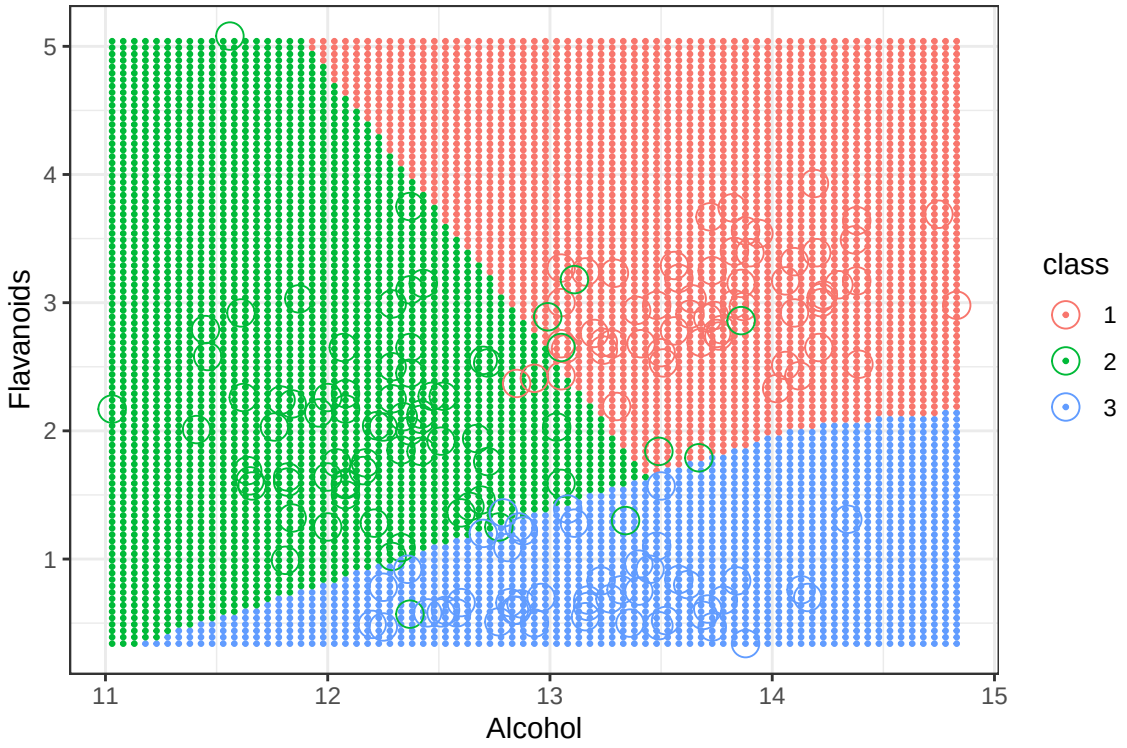
Rysunek 6: Obszary decyzyjne dla $C = 0.1$



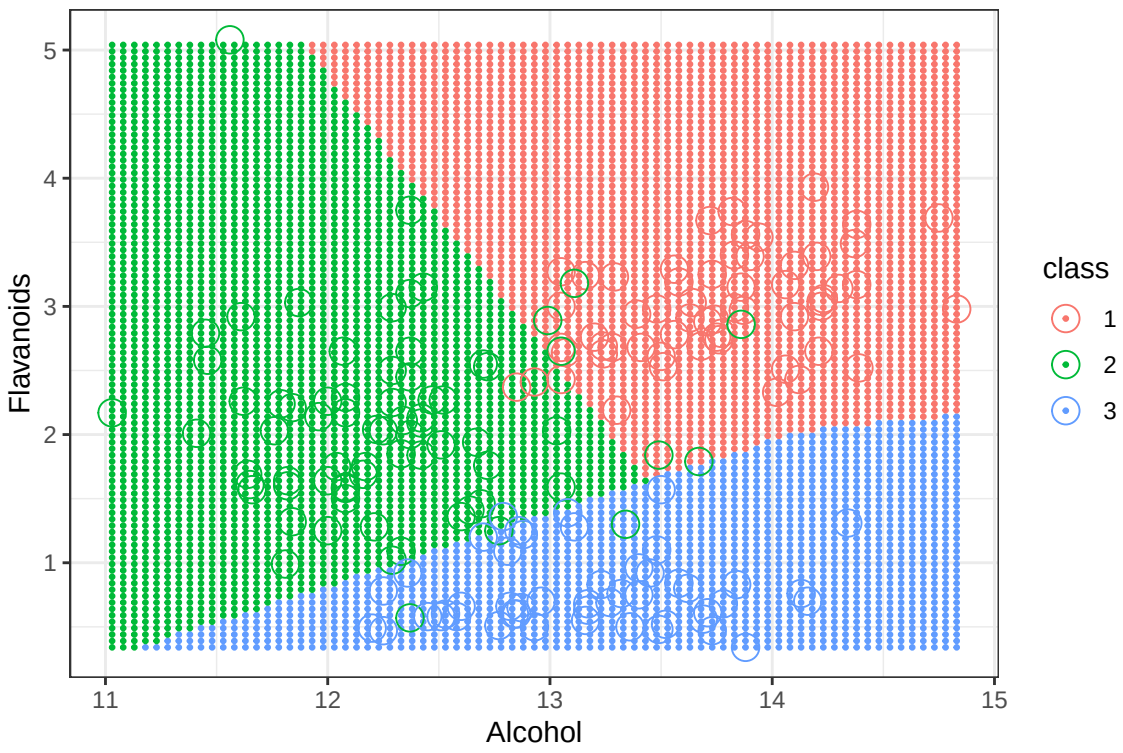
Rysunek 7: Obszary decyzyjne dla $C = 1$



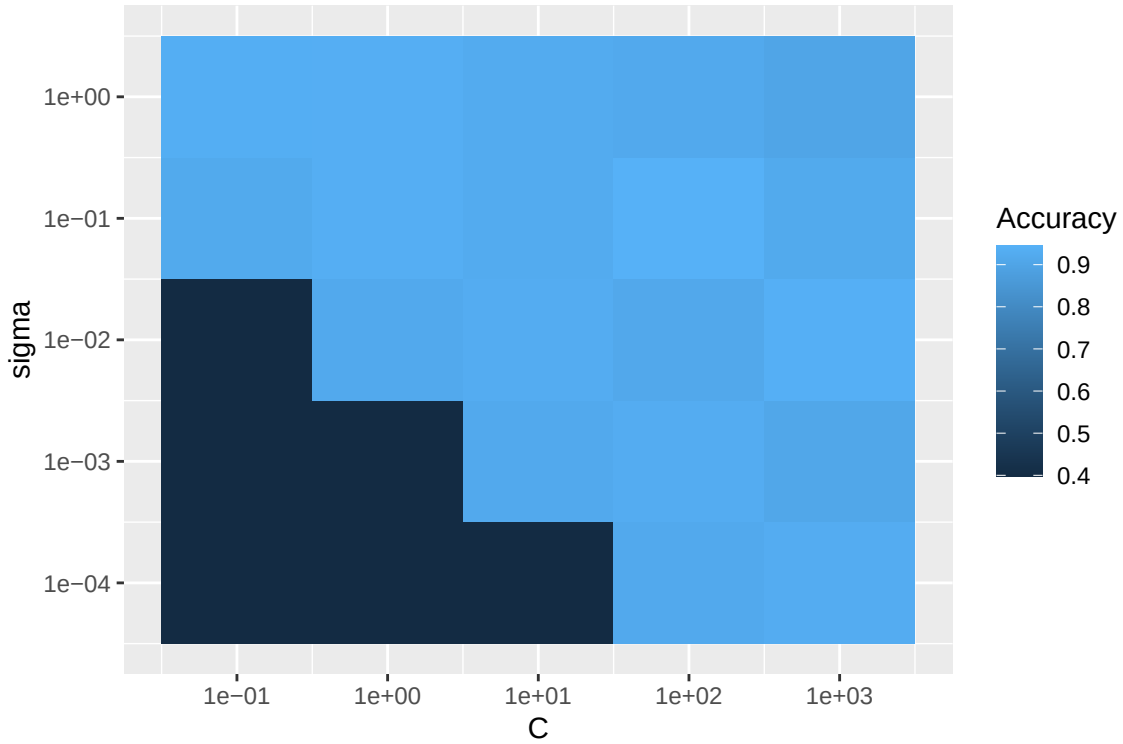
Rysunek 8: Obszary decyzyjne dla $C = 10$



Rysunek 9: Obszary decyzyjne dla $C = 100$



Rysunek 10: Obszary decyzyjne dla $C = 1000$



Rysunek 11: Mapa ciepła dokładności klasyfikatora

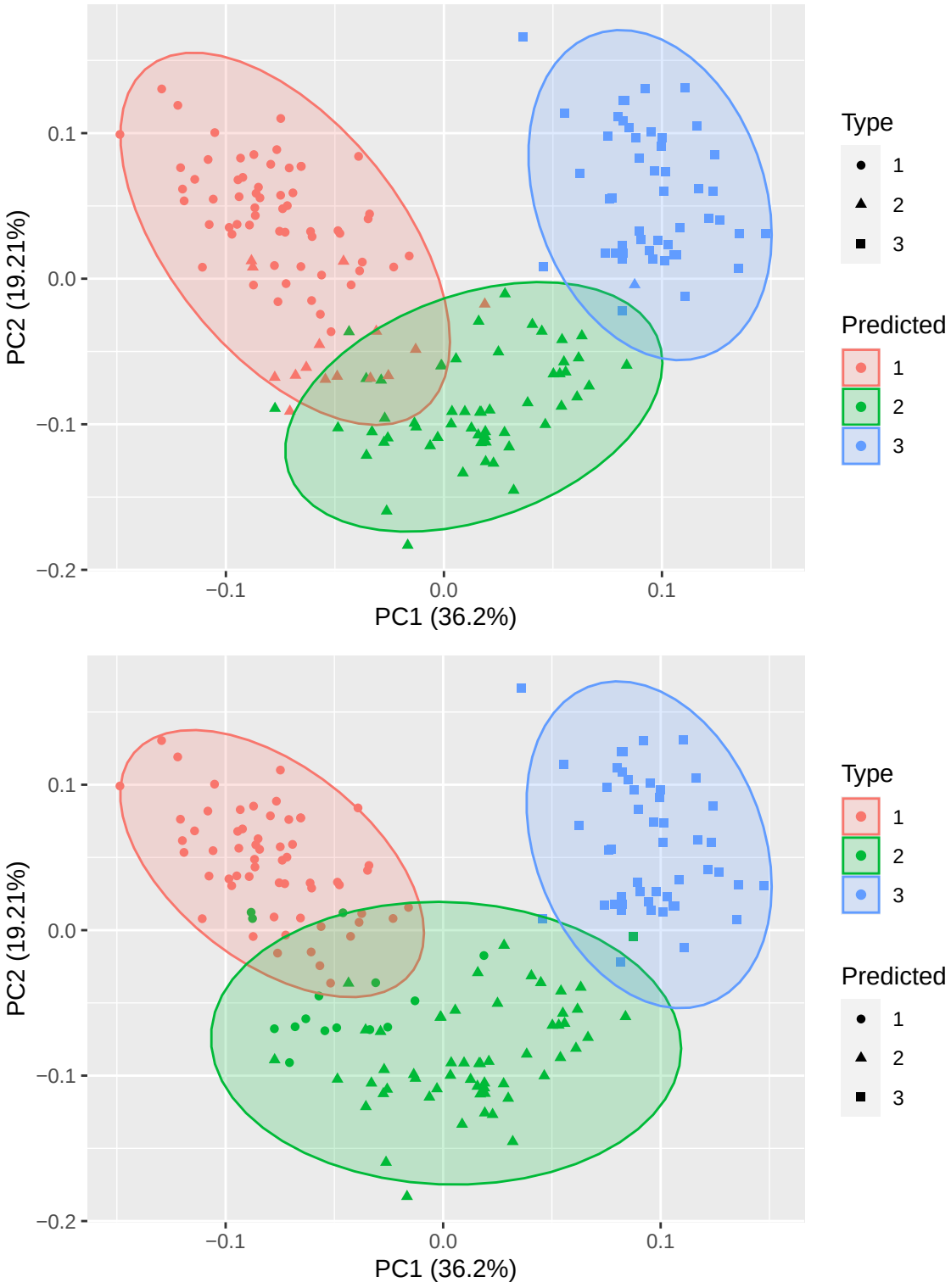
linear	polynomial	radial
0.928	0.933	0.938

Tabela 3: Porównanie klasyfikatorów dla różnych jąder

sigma	C
0.10	100.00

Tabela 4: Parametry dla najlepszego klasyfikatora

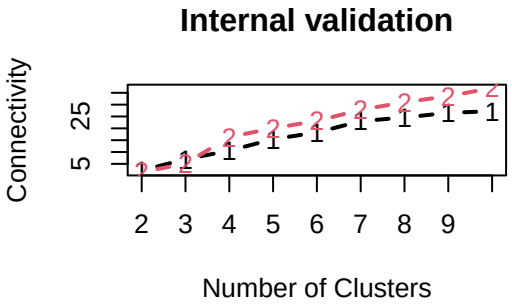
3 Zadanie 2

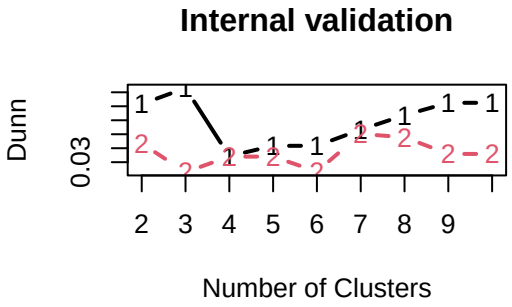


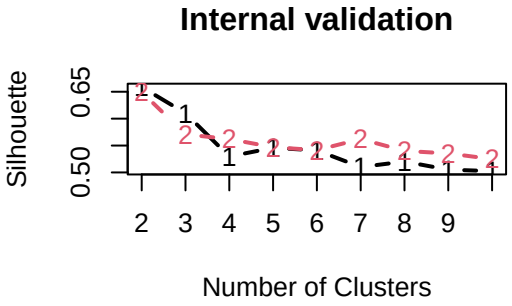

```
## Warning in clValid(wine %>% select(-Type), nClust = cl.range, clMethods =
## cl.methods, : rownames for data not specified, using 1:nrow(data)
```

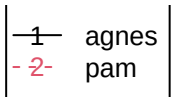
```
##
## Clustering Methods:
##  agnes pam
##
## Cluster sizes:
##  2 3 4 5 6 7 8 9 10
##
## Validation Measures:
##                            2       3       4       5       6       7       8       9
##
## agnes Connectivity   2.2329  6.9567 10.5615 15.4302 18.2468 23.0460 24.5746 26.5770 2
##       Dunn           0.0716  0.0830  0.0343  0.0417  0.0417  0.0532  0.0636  0.0725
##       Silhouette     0.6587  0.6101  0.5296  0.5458  0.5409  0.5101  0.5202  0.5051
## 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         0.0830 agnes  3
## Silhouette   0.6587 agnes  2
```

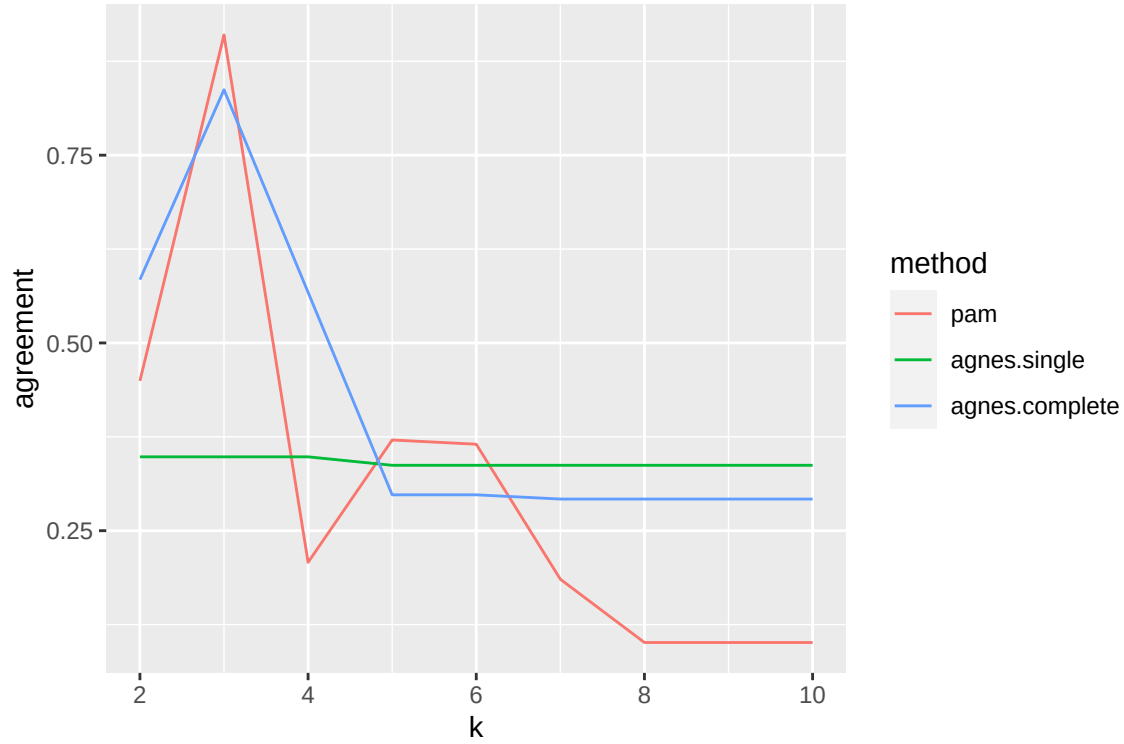
```
##                   Score Method Clusters
## Connectivity 1.52857143    pam        2
## Dunn         0.08304858  agnes        3
## Silhouette   0.65872930  agnes        2
```





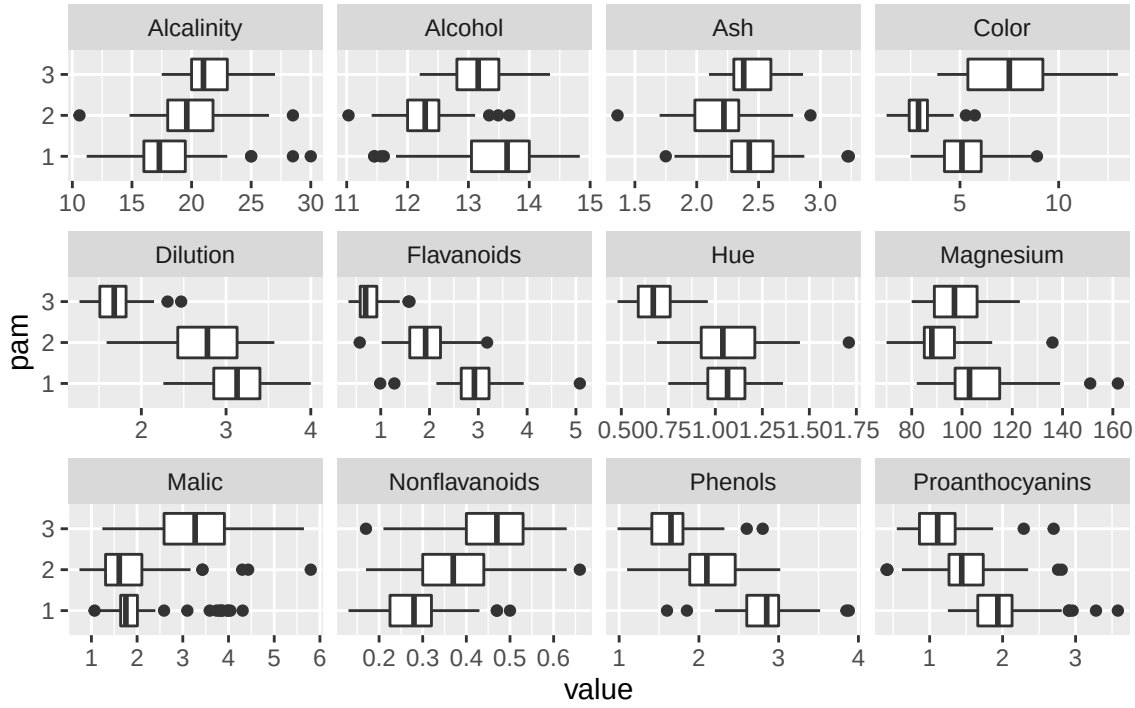




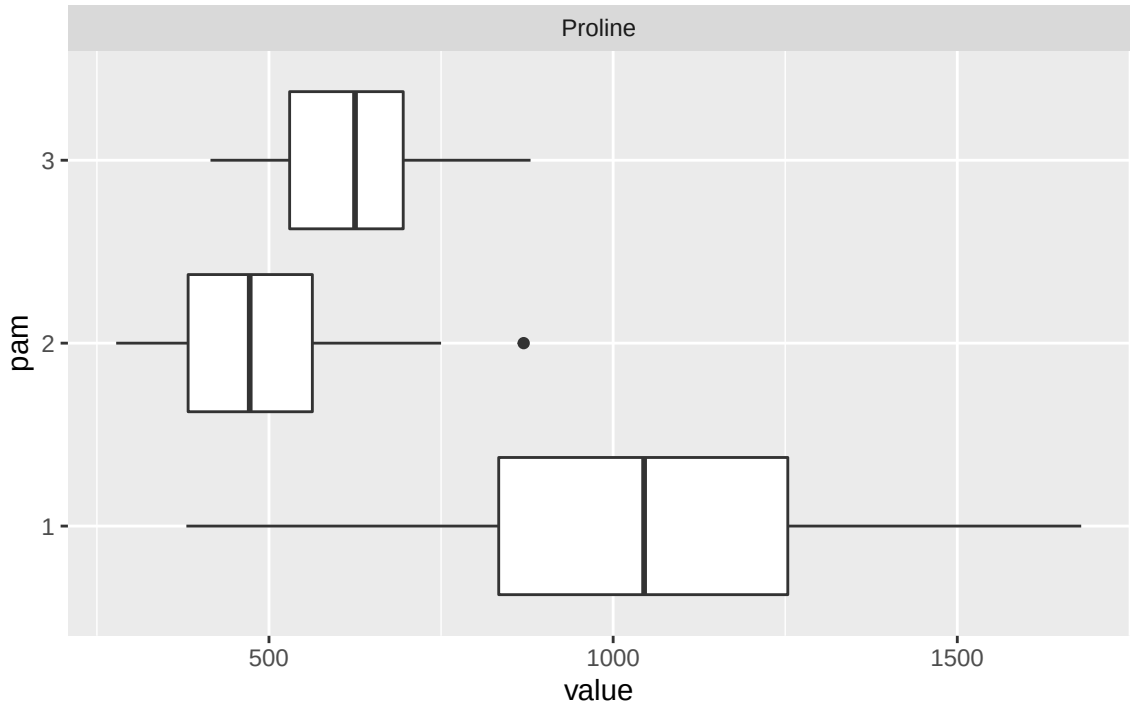



```
wine.pam <- pam(scale(wine %>% select(-Type)), 3)
wine.agnes <- cutree(agnes(scale(wine %>% select(-Type)), method='complete'), 3)
```

```
wine$pam <- as.factor(wine.pam$clustering)
wine$agnes <- as.factor(wine.agnes)

plot_boxplot(wine, by='pam')
```

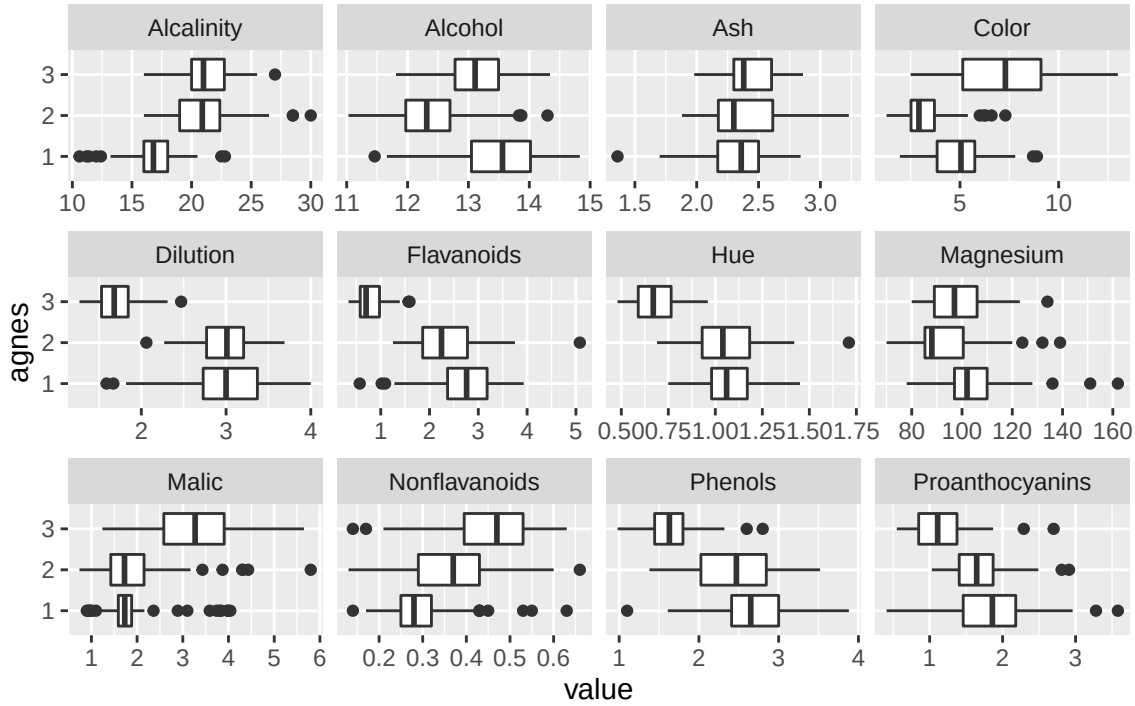


Page 1

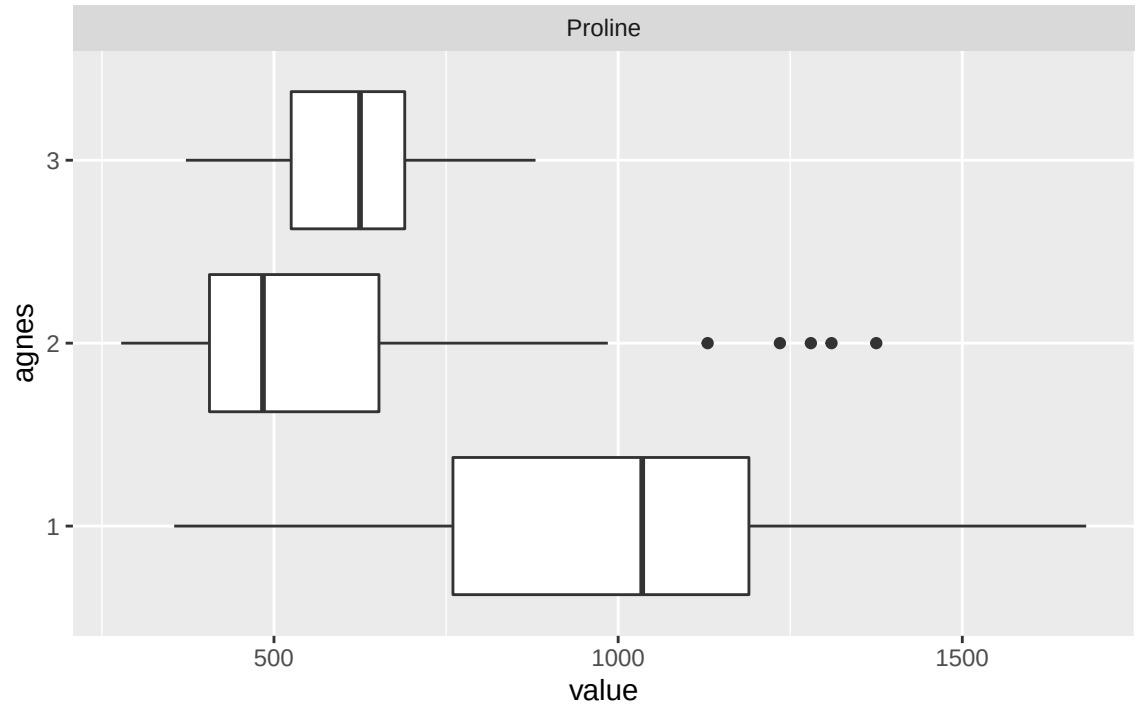


Page 2

```
plot_boxplot(wine, by='agnes')
```



Page 1




```
wine.pam$medoids
```

```
##         Alcohol      Malic         Ash Alcalinity   Magnesium    Phenols
## [1,]  0.5904981 -0.4711544  0.15849862  0.3009543  0.01809398  0.6469393
## [2,] -0.9246039 -0.5427655 -0.89856839 -0.1482061 -1.38222271 -1.0307762
## [3,]  0.3934117  0.8088930  0.04914686  0.6003946 -0.54203270 -0.5833854
##         Flavanoids Nonflavanoids Proanthocyanins       Color        Hue
## [1,]  0.9518166597   -0.81841060      0.47016154  0.01807806  0.3611585
## [2,]  0.0007311716    0.06545479      0.06831575 -0.71522236  0.1861586
## [3,] -1.2707199546    0.70826598     -0.59560339  1.45017064 -1.7825902
##        Dilution    Proline
## [1,]  1.2089101  0.5497067
## [2,]  0.7863692 -0.7522631
## [3,] -1.3967588 -0.3076880
```