Raport 4

Paweł Matławski
album 249732

14 marca 2021

Spis treści

1 Zaawansowane metody klasyfikacji

1.1 Rodziny klasyfikatorów

1.1.1 Wczytanie danych

Na początek wczytamy dane, z którymi pracowaliśmy w poprzednim sprawozdaniu. W naszym przypadku była to ramka **Vehicle** z biblioteki **mlbench**. Najpierw zastosujemy standaryzację, a następnie podzielimy dane na zbiór uczący i testowy, tak jak miało to miejsce w zadaniu 2 z listy 3.

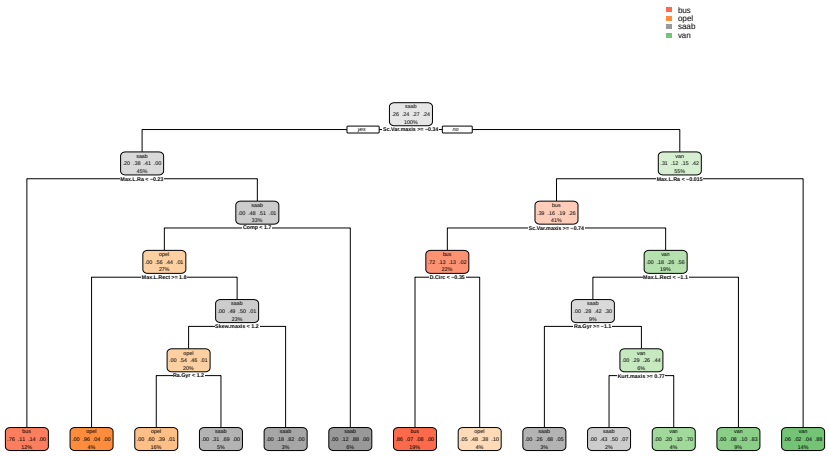
```
library(ipred)
library(rpart)
library(rpart.plot)
library(mlbench)
library(randomForest)
```

```
data("Vehicle")
attach(Vehicle)

set.seed(1)
veh1 <- scale(Vehicle[1:18])
veh1 <- data.frame(veh1)
etykietki.veh <- Vehicle$Class
etykietki.veh <- data.frame(etykietki.veh)
veh2 <- cbind(veh1, etykietki.veh)

n.veh <- dim(veh2)[1]
learning.indx.veh <- sample(1:n.veh,2/3*n.veh)
learning.set.veh <- veh2[learning.indx.veh,]
test.set.veh <- veh2[-learning.indx.veh,]
n.veh.learning <- floor(2*n.veh/3)
n.veh.test <- n.veh - n.veh.learning

etykietki.learning.veh <- learning.set.veh$etykietki.veh
etykietki.test.veh <- test.set.veh$etykietki.veh

model <- etykietki.veh~ .
veh.tree.simple <- rpart(model, data=learning.set.veh)

# prognozy dla zbioru uczącego
pred.labels.learning <- predict(veh.tree.simple, newdata=learning.set.veh, type = "class

# prognozy dla zbioru testowego
pred.labels.test <- predict(veh.tree.simple, newdata=test.set.veh, type = "class")

# wyznaczenie prognozowanych prawdopodobieństw a posteriori
pred.probs.test <- predict(veh.tree.simple, newdata=test.set.veh, type = "prob")
```

1.1.2 Drzewo klasyfikacyjne

Na początek zastosujemy znaną nam już metodę drzewa klasyfikacyjnego. Posłuży nam ona jako punkt odniesienia przy porównywaniu skuteczności "nowych" algorytmów.

```
rpart.plot(veh.tree.simple)
```



Rysunek 1: Pojedyncze drzewo klasyfikacyjne - zbiór uczący

Wyznaczamy macierze pomyłek i liczymy błąd klasyfikacyjny.

```
conf.mat.learning <- table(pred.labels.learning, learning.set.veh$etykietki.veh)
print(xtable(conf.mat.learning, caption="Macierz pomyłek - zbiór uczący"))
```

	bus	opel	saab	van
bus	140	14	17	0
opel	1	85	43	3
saab	0	26	80	2
van	5	10	10	128

Tabela 1: Macierz pomyłek - zbiór uczący

Błąd klasyfikacyjny dla zbioru uczącego:

```
(error.rate.learning <- (nrow(learning.set.veh) - sum(diag(conf.mat.learning))) / nrow(l

## [1] 0.2322695
```

```
conf.mat.test <- table(pred.labels.test, test.set.veh$etykietki.veh)
print(xtable(conf.mat.test, caption="Macierz pomyłek - zbiór testowy"))
```

	bus	opel	saab	van
bus	68	8	12	0
opel	0	35	22	2
saab	1	27	26	5
van	3	7	7	59

Tabela 2: Macierz pomyłek - zbiór testowy

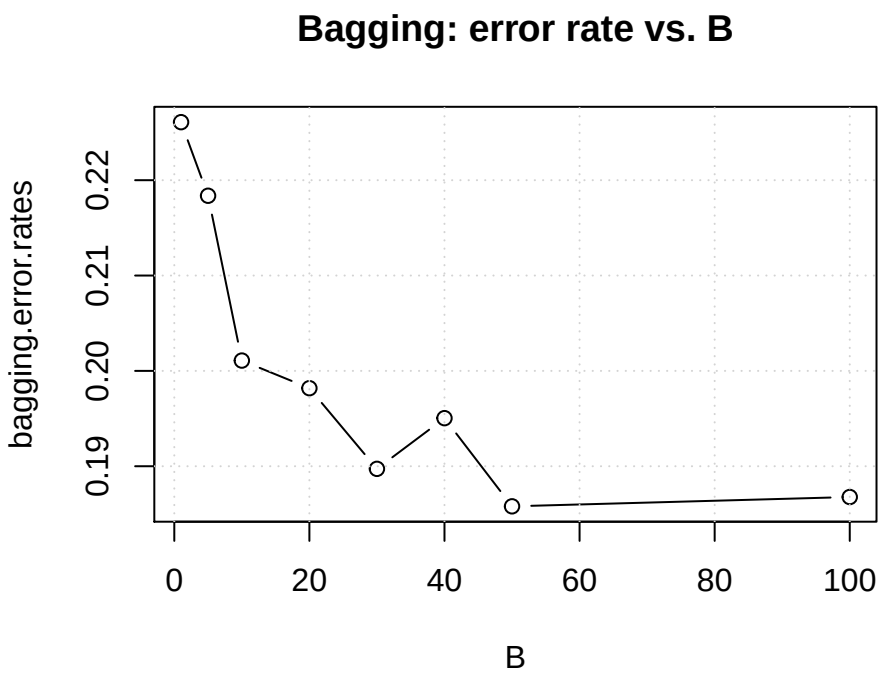
Błąd klasyfikacyjny dla zbioru testowego:

```
(error.rate.test <- (nrow(test.set.veh) - sum(diag(conf.mat.test))) / nrow(test.set.veh)

## [1] 0.3333333
```

1.1.3 Bagging

Pierwszą "nową" metodą, której użyjemy, będzie bagging (od angielskiego bootstrap aggregating). Na początek sprawdźmy, jak liczba replikacji wpływa na dokładność:

```
B.vector <- c(1, 5, 10, 20, 30, 40, 50, 100)
bagging.error.rates <- sapply(B.vector, function(b)  {errorest(Class~., data=Vehicle, mo
plot(B.vector, bagging.error.rates, xlab="B", main="Bagging: error rate vs. B", type="b"
grid()
```



Rysunek 2: Bagging - liczba replikacji a dokładność

Jak widać po wykresie, dokładność rośnie wraz ze wzrostem liczby replikacji, choć zdarzają się od tej reguły wyjątki. W naszej analizie użyjemy modelu z 50 replikacjami.

```
btree <- bagging(model, data=learning.set.veh, nbagg=50, minsplit=1, cp=0)
```

```
# prognozy dla zbioru uczącego
pred.labels.bagg.learning <- predict(btree, newdata=learning.set.veh, type = "class")

# prognozy dla zbioru testowego
pred.labels.bagg.test <- predict(btree, newdata=test.set.veh, type = "class")
```

Wyznaczamy macierze pomyłek i błąd klasyfikacyjny:

```
conf.mat.bagg.learning <- table(pred.labels.bagg.learning, learning.set.veh$etykietki.ve
print(xtable(conf.mat.bagg.learning, caption="Macierz pomyłek - zbiór uczący"))
```

	bus	opel	saab	van
bus	146	0	0	0
opel	0	134	0	0
saab	0	1	150	0
van	0	0	0	133

Tabela 3: Macierz pomyłek - zbiór uczący

Błąd klasyfikacyjny dla zbioru uczącego:

```
(error.rate.bagg.learning <- (nrow(learning.set.veh) - sum(diag(conf.mat.bagg.learning))

## [1] 0.00177305
```

```
conf.mat.bagg.test <- table(pred.labels.bagg.test, test.set.veh$etykietki.veh)
print(xtable(conf.mat.bagg.test, caption="Macierz pomyłek - zbiór testowy"))
```

	bus	opel	saab	van
bus	68	1	5	0
opel	1	40	21	0
saab	0	31	37	1
van	3	5	4	65

Tabela 4: Macierz pomyłek - zbiór testowy

Błąd klasyfikacyjny dla zbioru testowego:

```
(error.rate.bagg.test <- (nrow(test.set.veh) - sum(diag(conf.mat.bagg.test))) / nrow(tes

## [1] 0.2553191
```

Bagging spisuje się bardzo dobrze - błąd klasyfikacyjny dla zbioru uczącego w każdej próbie jest bliski 0. Nieco gorzej prezentuje się on w przypadku zbioru testowego, ale wciąż zauważalna jest poprawa względem pojedynczego drzewa klasyfikacyjnego.

1.1.4 Boosting

Kolejnym algorytmem, który wykorzystamy w naszym sprawozdaniu jest boosting. Działa on w ten sposób, że w kolejnych iteracjach trenuje a następnie mierzy błąd wszystkich dostępnych słabych klasyfikatorów. W każdej następnej iteracji "ważność" źle zakwalifikowanych obserwacji jest zwiększana, tak że klasyfikatory zwracają na nie większą uwagę (źródło: Wikipedia).

```
library(adabag)
boost <- boosting(model, data=learning.set.veh, mfinal = 15)
```

```
# prognozy dla zbioru uczącego
pred.labels.boost.learning <- as.factor(predict.boosting(boost, newdata=learning.set.veh

# prognozy dla zbioru testowego
pred.labels.boost.test <- as.factor(predict.boosting(boost, newdata=test.set.veh)$class)
```

Wyznaczamy macierze pomyłek i błąd klasyfikacyjny:

```
conf.mat.boost.learning <- table(pred.labels.boost.learning, learning.set.veh$etykietki.
print(xtable(conf.mat.boost.learning, caption="Macierz pomyłek - zbiór uczący"))
```

	bus	opel	saab	van
bus	146	0	0	0
opel	0	133	2	0
saab	0	1	147	0
van	0	1	1	133

Tabela 5: Macierz pomyłek - zbiór uczący

Błąd klasyfikacyjny dla zbioru uczącego:

```
(error.rate.boost.learning <- (nrow(learning.set.veh) - sum(diag(conf.mat.boost.learning

## [1] 0.008865248
```

```
conf.mat.boost.test <- table(pred.labels.boost.test, test.set.veh$etykietki.veh)
print(xtable(conf.mat.boost.test, caption="Macierz pomyłek - zbiór testowy"))
```

Błąd klasyfikacyjny dla zbioru testowego:

	bus	opel	saab	van
bus	66	0	1	0
opel	2	35	22	0
saab	2	36	42	2
van	2	6	2	64

Tabela 6: Macierz pomyłek - zbiór testowy

```
(error.rate.boost.test <- (nrow(test.set.veh) - sum(diag(conf.mat.boost.test))) / nrow(t

## [1] 0.2659574
```

Jak widać algorytm boosting działa dobrze nawet przy niewielkiej liczbie iteracji (w naszej analizie było to 15). Jest on jednak bardzo czasochłonny - stąd właśnie niewielka liczba iteracji.

1.1.5 Lasy losowe

Ostatnią z metod przetestowanych przez nas w tym zadaniu są lasy losowe (ang. random forest).

```
p <-  ncol(Vehicle) - 1
las <- randomForest(model, data=learning.set.veh, ntree=100, mtry=sqrt(p), importance=TR
```

```
# prognozy dla zbioru uczącego
pred.labels.las.learning <- predict(las, newdata=learning.set.veh, type="class")

# prognozy dla zbioru testowego
pred.labels.las.test <- predict(las, newdata=test.set.veh, type="class")
```

Wyznaczamy macierze pomyłek i błąd klasyfikacyjny:

```
conf.mat.las.learning <- table(pred.labels.las.learning, learning.set.veh$etykietki.veh)
print(xtable(conf.mat.las.learning, caption="Macierz pomyłek - zbiór uczący"))
```

	bus	opel	saab	van
bus	146	0	0	0
opel	0	135	0	0
saab	0	0	150	0
van	0	0	0	133

Tabela 7: Macierz pomyłek - zbiór uczący

Błąd klasyfikacyjny dla zbioru uczącego:

```
(error.rate.las.learning <- (nrow(learning.set.veh) - sum(diag(conf.mat.las.learning)))

## [1] 0
```

```
conf.mat.las.test <- table(pred.labels.las.test, test.set.veh$etykietki.veh)
print(xtable(conf.mat.las.test, caption="Macierz pomyłek - zbiór testowy"))
```

	bus	opel	saab	van
bus	69	0	3	0
opel	0	40	26	0
saab	0	31	34	1
van	3	6	4	65

Tabela 8: Macierz pomyłek - zbiór testowy

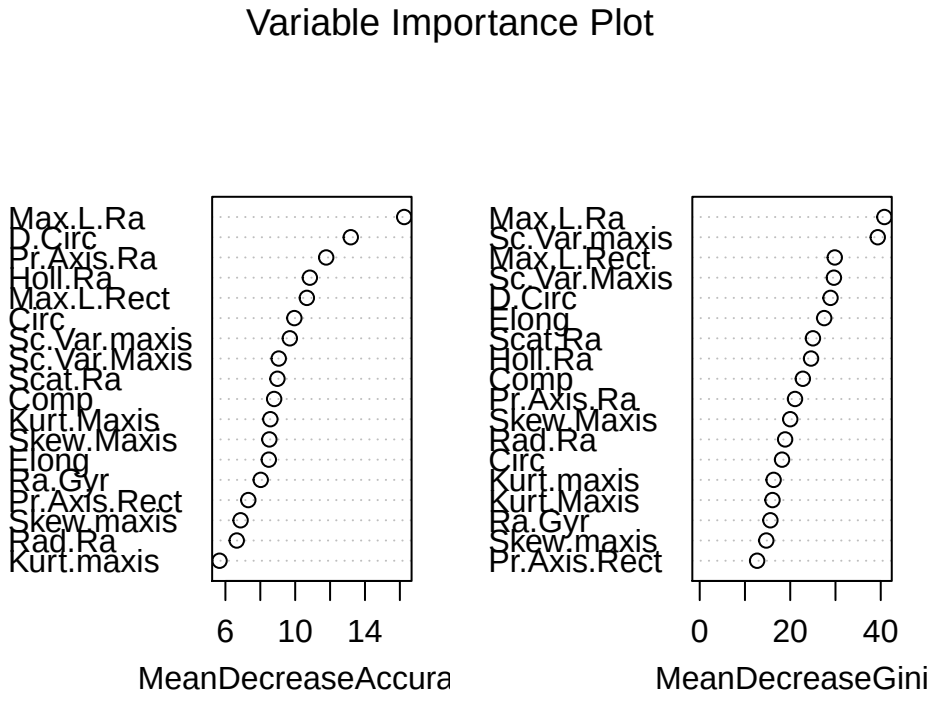
Błąd klasyfikacyjny dla zbioru testowego:

```
(error.rate.las.test <- (nrow(test.set.veh) - sum(diag(conf.mat.las.test))) / nrow(test.

## [1] 0.2624113
```

Rezultaty w przypadku tego algorytmu są bardzo dobrze. Działa on też nieco szybciej niż chociażby boosting.

1.1.6 Ważność cech

```
# Ranking ważności cech
varImpPlot(las, main = "Variable Importance Plot")
```



Rysunek 3: Wykres ważności cech

Zmiennej o największej zdolności dyskryminacyjnej jest Max.L.Ra, a o najmniejszej Skew.maxis, Pr.Axis.Rect oraz Kurt.maxis. Oznacza to, że zmienne wybrane przez nas w zadniu drugim z listy 3 nie były wcale tymi, które mają największy wpływ na budowę skutecznego modelu.

1.1.7 Podsumowanie

- Wszystkie trzy algorytmy dają wyraźnie lepsze rezulataty niż pojedyncze drzewo klasyfikacyjne.
- Różnice w błedach klasyfikacyjnych w przypadku trzech testowanych metod nie są duże, wszystkie dają zbliżone wyniki.
- Zdecydowanie najwolniejszą i najbardziej skomplikowaną metodą jest boosting. Na drugim miejscu znajduje się bagging, a najszybsze okazały się być lasy losowe.
- Nie wszystkie zmienne mają takie same zdolności dyskryminacyjne, jednak stworzenie skutecznego modelu wymaga użycia większości z nich (a najlepiej wszystkich).

1.2 Metoda wektorów nośnych (SVN)

Głównym celem tej metody jest segregacja zbioru w taki sposób, by odległość między najbliższymi punktami (margines) była możliwie jak największa. W teorii metoda ta powinna cechować się sporą dokładnością.

1.2.1 Jądro liniowe

Zbudujemy klasyfikator SVM dla jądra liniowego i postaramy się zbadać, jak wartość C (parametr kosztu) wpływa na dokładność klasyfikacji.

- C=0.1

  Liczba wektorów nośnych w klasach:

```
library(e1071)
svm.linear.C0.1 <- svm(model, data=learning.set.veh, kernel="linear", cost=.1)
svm.linear.C0.1$nSV

## [1]  73  66 128 126
```

  Dokładność klasyfikacji:

```
pred.svm.lin.C0.1 <- predict(svm.linear.C0.1, newdata=test.set.veh)
(acc.svm.lin.C0.1   <- sum(diag(table(pred.svm.lin.C0.1, etykietki.test.veh)))/n.ve

## [1] 0.7340426
```

- C=1

  Liczba wektorów nośnych w klasach:

```
svm.linear.C1 <- svm(model, data=learning.set.veh, kernel="linear", cost=1)
svm.linear.C1$nSV

## [1]  41  34 113 110
```

Dokładność klasyfikacji:

```
pred.svm.lin.C1 <- predict(svm.linear.C1, newdata=test.set.veh)
(acc.svm.lin.C1   <- sum(diag(table(pred.svm.lin.C1, etykietki.test.veh)))/n.veh.te

## [1] 0.751773
```

- C=5

  Liczba wektorów nośnych w klasach:

```
svm.linear.C5 <- svm(model, data=learning.set.veh, kernel="linear", cost=5)
svm.linear.C5$nSV

## [1]  29  23 102 104
```

  Dokładność klasyfikacji:

```
pred.svm.lin.C5 <- predict(svm.linear.C5, newdata=test.set.veh)
(acc.svm.lin.C5   <- sum(diag(table(pred.svm.lin.C5, etykietki.test.veh)))/n.veh.te

## [1] 0.7659574
```

- C=10

  Liczba wektorów nośnych w klasach:

```
svm.linear.C10 <- svm(model, data=learning.set.veh, kernel="linear", cost=10)
svm.linear.C10$nSV

## [1]  23  16  97 100
```

  Dokładność klasyfikacji:

```
pred.svm.lin.C10 <- predict(svm.linear.C10, newdata=test.set.veh)
(acc.svm.lin.C10   <- sum(diag(table(pred.svm.lin.C10, etykietki.test.veh)))/n.veh.

## [1] 0.7836879
```

- C=50

  Liczba wektorów nośnych w klasach:

```
svm.linear.C50 <- svm(model, data=learning.set.veh, kernel="linear", cost=50)
svm.linear.C50$nSV

## [1] 21 15 96 98
```

Dokładność klasyfikacji:

```
pred.svm.lin.C50 <- predict(svm.linear.C50, newdata=test.set.veh)
(acc.svm.lin.C50   <- sum(diag(table(pred.svm.lin.C10, etykietki.test.veh)))/n.veh.

## [1] 0.7836879
```

Dokładność klasyfikacji rośnie wraz ze wzrostem parametru C i waha się między 73-78%. Zauważamy ponadto, że im wyższy parametr kosztu, tym mniej wektorów nośnych w klasach.

1.2.2 SVM dla innych funkcji jądrowych

Zbudujemy klasyfikator dla jądra wielomianowego 2 i 4 stopnia, a także jądra radialnego.

```
svm.poly2  <- svm(model, data=learning.set.veh, kernel="polynomial", degree = 2)
svm.poly4  <- svm(model, data=learning.set.veh, kernel="polynomial", degree = 4)
svm.radial <- svm(model, data=learning.set.veh, kernel="radial")

pred.svm.poly2  <- predict(svm.poly2, newdata=test.set.veh)
pred.svm.poly4  <- predict(svm.poly4, newdata=test.set.veh)
pred.svm.radial <- predict(svm.radial, newdata=test.set.veh)
```

Dokładność klasyfikacji:

- Jądro wielomianowe stponia 2

```
(acc.svm.poly2  <- sum(diag(table(pred.svm.poly2, etykietki.test.veh)))/n.veh.test)

## [1] 0.6170213
```

- Jądro wielomianowe stponia 4

```
(acc.svm.poly4  <- sum(diag(table(pred.svm.poly4, etykietki.test.veh)))/n.veh.test)

## [1] 0.5425532
```

- Jądro radialne

```
(acc.svm.radial <- sum(diag(table(pred.svm.radial, etykietki.test.veh)))/n.veh.test

## [1] 0.7375887
```

Dokładność klasyfikacji w przypadku jąder wielomianowych nie jest zbyt wysoka. W przypadku jądra radialnego jest ona porównywalna do jądra liniowego z niskim parametrem kosztu.

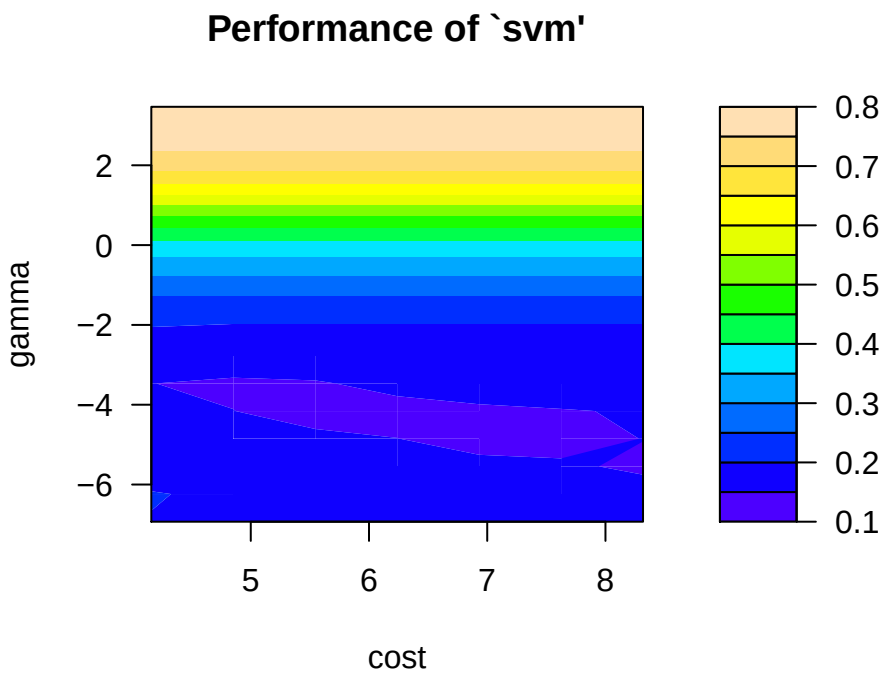
1.2.3 Optymalizacja parametrów

Postaramy się "dostroić" jednocześnie parametry C oraz γ.

```
C.range <- 2^(6:12)
gamma.range <- 2^((-10):5)
radial.tune <- tune(svm, train.x=model,
                    data = learning.set.veh,
                    kernel="radial",
                    ranges=list(cost=C.range, gamma=gamma.range))
radial.tune$best.parameters

##    cost    gamma
## 32  512 0.015625
```

Zobrazujmy zależności błędu od parametrów jądra:

```
plot(radial.tune, transform.x=log, transform.y=log, color.palette = topo.colors)
```



Rysunek 4: Jądro radialne - błąd a parametry

Sprawdźmy teraz, jak dokładny jest model z optymalnie dobranymi parametrami:

```
C.best <- radial.tune$best.parameters[["cost"]]
gamma.best <- radial.tune$best.parameters[["gamma"]]

svm.radial.tuned <- svm(model,
                        data=learning.set.veh, kernel="radial",
                        cost=C.best, gamma=gamma.best)

pred.svm.radial.tuned <- predict(svm.radial.tuned, newdata=test.set.veh)
(acc.svm.radial <- sum(diag(table(pred.svm.radial.tuned, etykietki.test.veh)))/n.veh.tes

## [1] 0.8156028
```

Dokładność przekroczyła 81%, co należy uznać za bardzo dobry wynik. Jest to wyraźna poprawa względem nieoptymalnie dobranych parametrów.

1.2.4 Podsumowanie

- Jądro radialne daje dużo lepsze wyniki od jądra wielomianowego.
- Dzięki optymalizacji parametrów możemy znacznie poprawić dokładność klasyfikacji.
- Zadowalające rezultaty udało nam się również uzyskać w przypadku jądra liniowego. Tam również dokładność można poprawić optymalnie dobierając parametr C.

2 Zadanie nr 2. Analiza skupień

2.1 Wybór danych

Raz jeszcze analizować będziemy zbiór **Vehicle** z biblioteki **mlbench**. Celem zadania jest wykorzystanie algorytmów analizy skupień, sprawdzenia ich efektywności i porównania metod. Przeprowadzimy analizę wykorzystując jeden algorytm grupujący oraz jeden algorytm hierarchiczny.

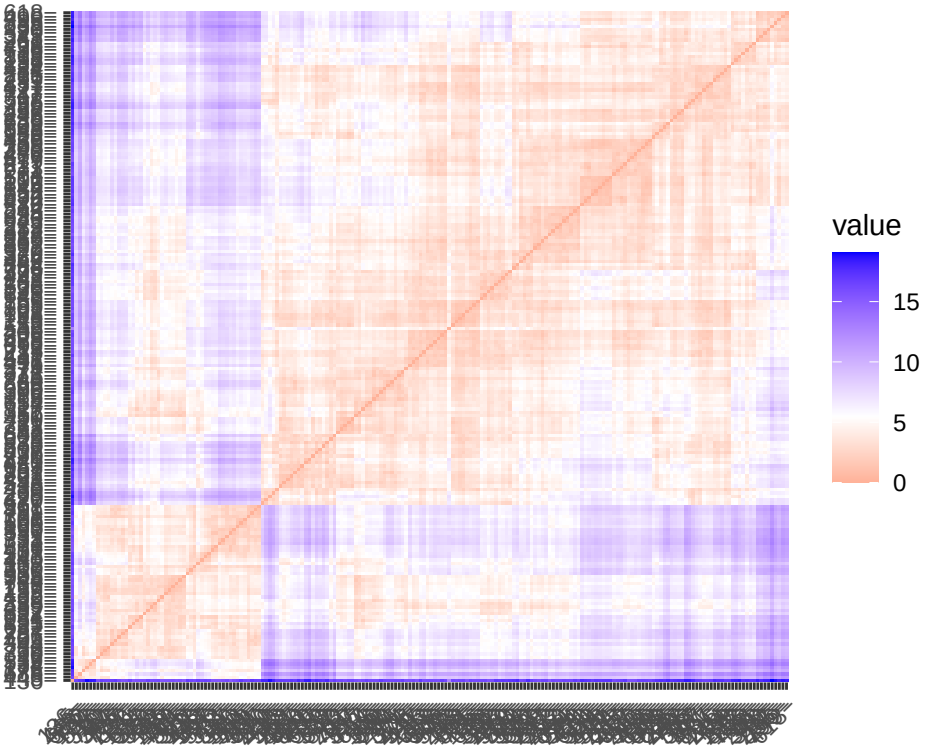
```
Vehicle2 <- Vehicle[sample(nrow(Vehicle), 200),]
Vehicle3 <- Vehicle2[,-19]
etykietki.veh <- Vehicle2$Class

veh1 <- as.data.frame((scale(Vehicle3[1:18])))
```

2.2 Zastosowanie algorytmu grupującego PAM

Jako pierwszy zastosujemy algorytm grupujący PAM (Partitioning Around Medoids. Przyjmuję liczbę skupień równą aktualnej liczbie klas, czyli K=4).

2.2.1 wizualizacja macierzy niepodobieństwa

```
library(MASS)
library(cluster)
vehicles.macniepod <- daisy(veh1)
vehicles.macniepod.matrix <- as.matrix(vehicles.macniepod)
library(factoextra)
fviz_dist(vehicles.macniepod, order = TRUE)
```

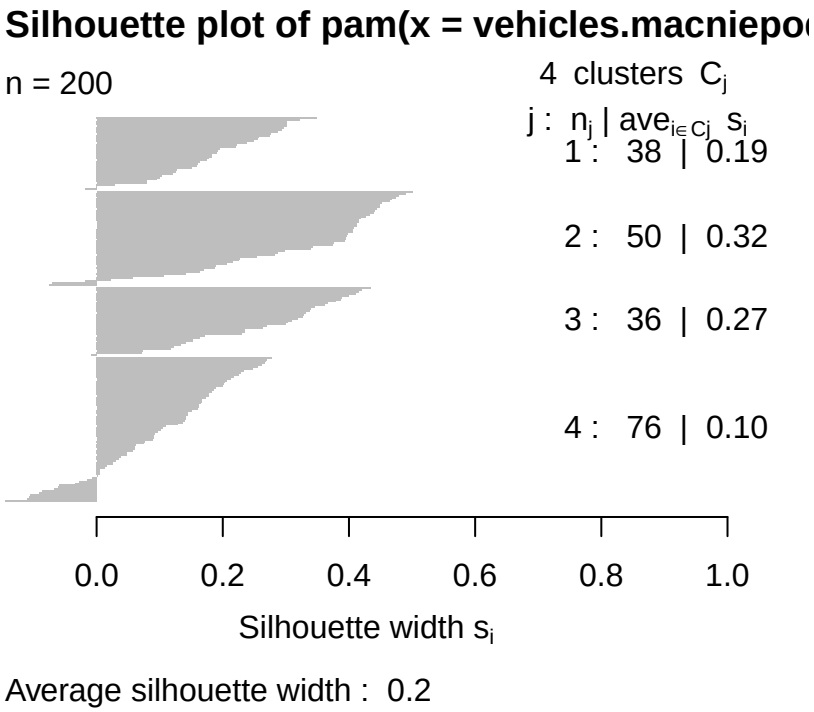


Rysunek 5: Macierz niepodobieństwa

2.2.2 zastosowanie metody oraz wizualizacja wyników

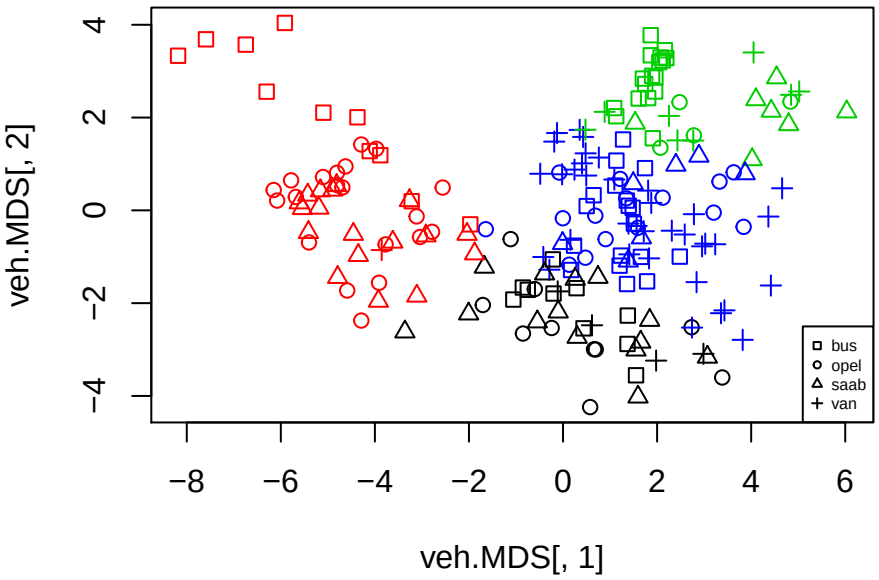
Stosujemy algorytm PAM, a następnie przedstawiamy domyślną wizualizację. Później poprzez metodę MDS redukujemy wymiar danych i przedstawiamy wizualizację metody w 2D.

```
Vehicles.pam <- pam(x=vehicles.macniepod, diss=TRUE, k=4)
etykietki.pam <- Vehicles.pam$clustering

#wizualizacja domyślna
plot(Vehicles.pam)
```



Rysunek 6: Domyślna wizualizacja

```
#wizualizacja wyników w 2D MDS
veh.MDS <- cmdscale(d=vehicles.macniepod, k=4)
symbole <- 0:4
plot(veh.MDS[,1], veh.MDS[,2], col=etykietki.pam, pch=symbole[Vehicle2$Class])
legend(x="bottomright",cex = 0.5, pch=symbole, legend=levels(Vehicle2$Class))
```



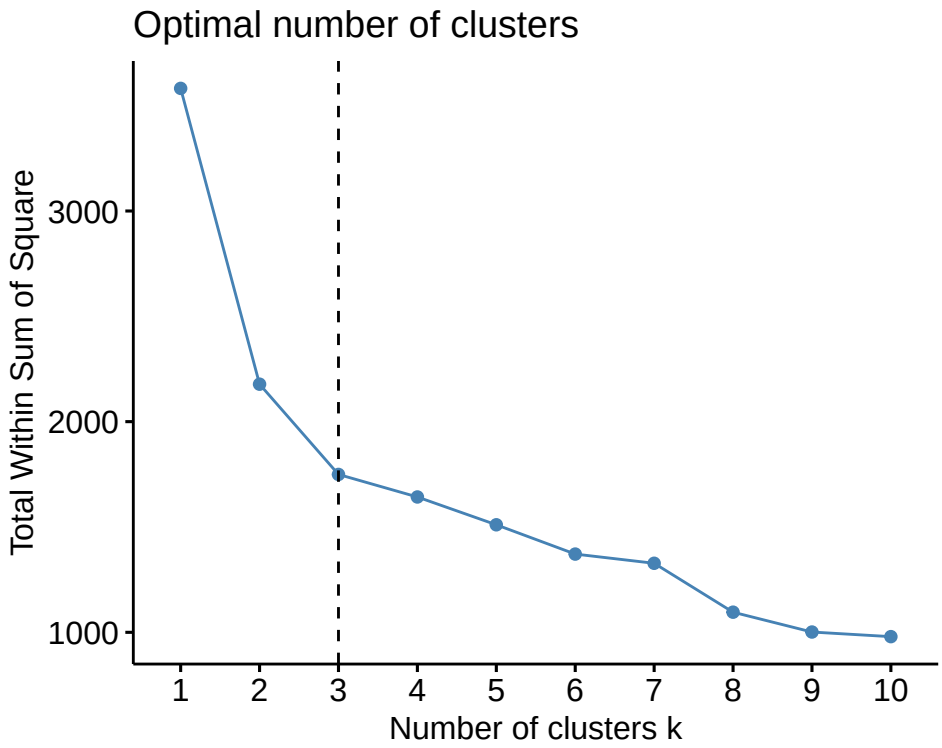
Rysunek 7: Wizualizacja w 2D

Zdecydowanie widać po wizualizacji, że algorytm PAM nie grupuje danych zgodnie z rzeczywistymi wartościami. Prawie w każdym z czterech skupisk możemy zaobserwować dużo elementów z różnych klas pierwotnych.

2.2.3 Wybór optymalnej liczby klastrów K

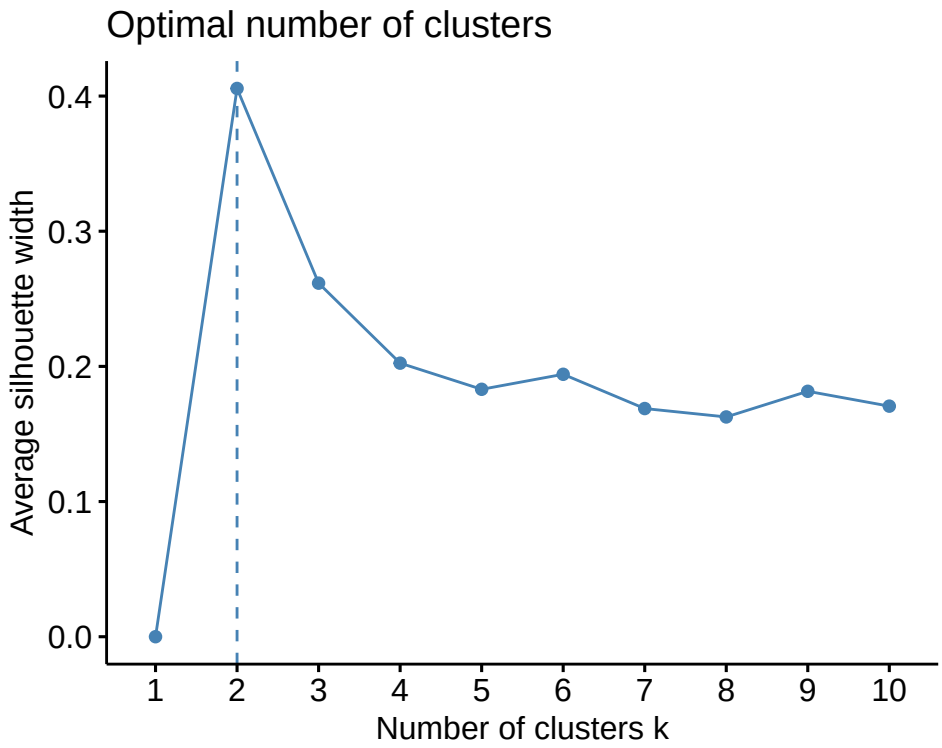
Dokonamy wyboru ilości skupień, aby algorytm mógł najbardziej optymalnie podzielić nasze dane na wyodrębnione grupy.

```
fviz_nbclust(veh1, FUNcluster = pam, method = "wss") + geom_vline(xintercept = 3, linety
```



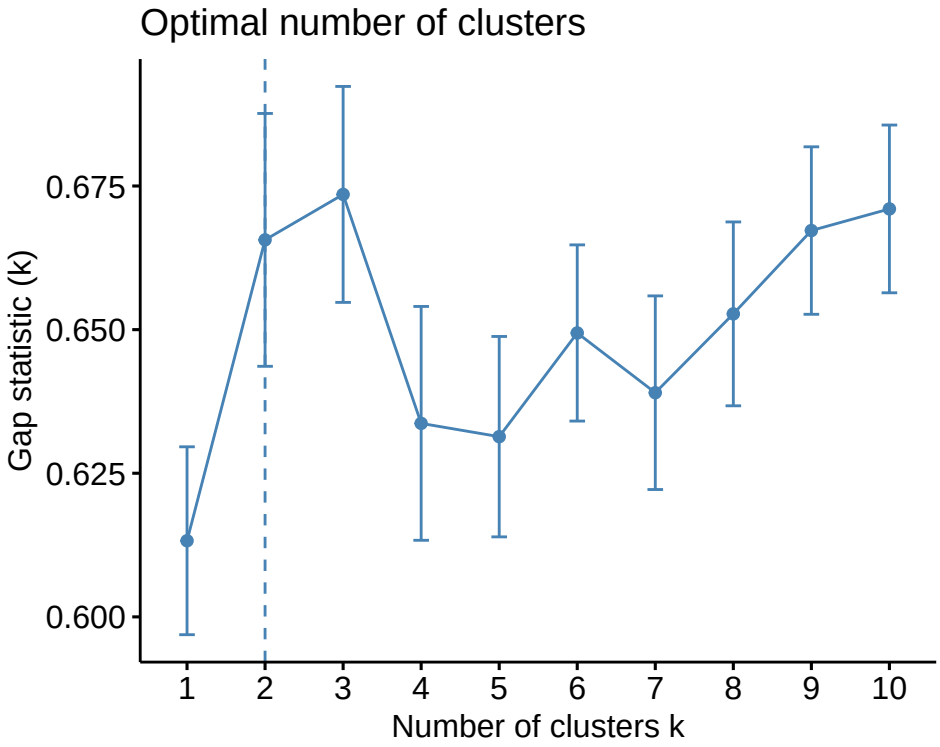
Rysunek 8: Wybór optymalnej liczby klastrów

```
fviz_nbclust(veh1, FUNcluster = pam, method = "silhouette")
```



Rysunek 9: Wybór optymalnej liczby klastrów

```
fviz_nbclust(veh1, FUNcluster = pam, method = "gap")
```



Rysunek 10: Wybór optymalnej liczby klastrów

Na podstawie trzech różnych metod znajdowania optymalnego K (”within cluster sums of squares”, ”average silhouette”, ”gap statistics”) decydujemy się na K=2.

Zastosujmy algorytm dla innych K niż początkowo i sprawdźmy średnicę, rozmiar oraz separację dla K=2,3,4,5,6.

- Dla K=2

```
Vehicles.pam2 <- pam(x=vehicles.macniepod, diss=TRUE, k=2)
Vehicles.pam3 <- pam(x=vehicles.macniepod, diss=TRUE, k=3)
Vehicles.pam5 <- pam(x=vehicles.macniepod, diss=TRUE, k=5)
Vehicles.pam6 <- pam(x=vehicles.macniepod, diss=TRUE, k=6)

sep.k2<-as.data.frame(Vehicles.pam2$clusinfo[,-c(2,3)])
print(xtable(sep.k2,caption="Średnica, rozmiar i separacja dla K = 2"),table.placem
```

	size	diameter	separation
1	149.00	10.15	1.42
2	51.00	18.92	1.42

Tabela 9: Średnica, rozmiar i separacja dla K = 2

- Dla K=3

```
sep.k3<-as.data.frame(Vehicles.pam3$clusinfo[,-c(2,3)])
print(xtable(sep.k3,caption="Średnica, rozmiar i separacja dla K = 3"),table.placem
```

	size	diameter	separation
1	76.00	7.19	1.42
2	51.00	18.92	1.42
3	73.00	7.80	1.51

Tabela 10: Średnica, rozmiar i separacja dla K = 3

- Dla K=4

```
sep.k4<-as.data.frame(Vehicles.pam$clusinfo[,-c(2,3)])
print(xtable(sep.k4,caption="Średnica, rozmiar i separacja dla K = 4"),table.placem
```

	size	diameter	separation
1	38.00	7.95	1.27
2	50.00	18.92	1.42
3	36.00	7.08	1.00
4	76.00	7.35	1.00

Tabela 11: Średnica, rozmiar i separacja dla K = 4

- Dla K=5

```
sep.k5<-as.data.frame(Vehicles.pam5$clusinfo[,-c(2,3)])
print(xtable(sep.k5,caption="Średnica, rozmiar i separacja dla K = 5"),table.placem
```

	size	diameter	separation
1	34.00	17.12	1.20
2	36.00	7.93	1.67
3	36.00	7.08	1.00
4	69.00	6.34	1.00
5	25.00	6.34	1.42

Tabela 12: Średnica, rozmiar i separacja dla K = 5

- Dla K=6

```
sep.k6<-as.data.frame(Vehicles.pam6$clusinfo[,-c(2,3)])
print(xtable(sep.k6,caption="Średnica, rozmiar i separacja dla K = 6"),table.placem
```

	size	diameter	separation
1	27.00	16.73	0.98
2	36.00	7.93	1.67
3	40.00	5.73	1.48
4	39.00	5.67	1.44
5	18.00	5.45	1.44
6	40.00	5.47	0.98

Tabela 13: Średnica, rozmiar i separacja dla K = 6

A więc, wyświetlone dane potwierdzają nam, że podział na dwa skupienia jest najbardziej optymalny.

2.2.4 Porównanie wyników klasyfikacji z rzeczywistymi klasami

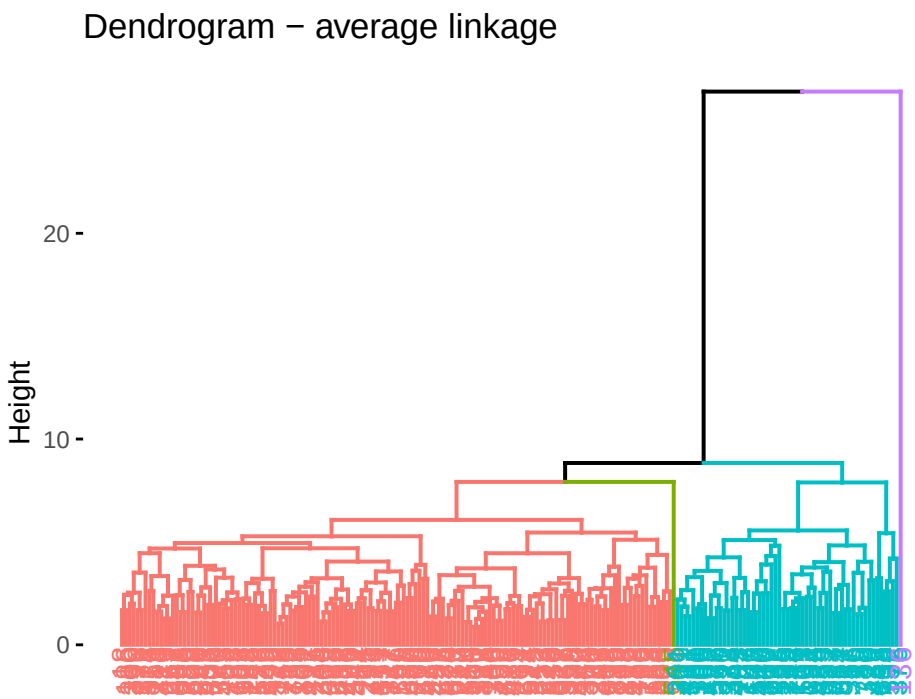
```
library(e1071)
etykietki.pam2 <- Vehicles.pam2$clustering
etykietki.pam3 <- Vehicles.pam3$clustering
etykietki.pam5 <- Vehicles.pam5$clustering
etykietki.pam6 <- Vehicles.pam6$clustering

com2<-table(etykietki.pam2,etykietki.veh)
diag2<-compareMatchedClasses(etykietki.pam2,etykietki.veh,method = "rowmax")$diag
cat("Skuteczność Pam dla K=2:",diag2)

## Skuteczność Pam dla K=2: 0.6938776

com3<-table(etykietki.pam3,etykietki.veh)
diag3<-compareMatchedClasses(etykietki.pam3,etykietki.veh,method = "rowmax")$diag
cat("Skuteczność Pam dla K=3:",diag3)

## Skuteczność Pam dla K=3: 0.4805195

com4<-table(etykietki.pam,etykietki.veh)
diag4<-compareMatchedClasses(etykietki.pam,etykietki.veh,method = "rowmax")$diag
cat("Skuteczność Pam dla K=4:",diag4)

## Skuteczność Pam dla K=4: 0.435

com5<-table(etykietki.pam5,etykietki.veh)
diag5<-compareMatchedClasses(etykietki.pam5,etykietki.veh,method = "rowmax")$diag
cat("Skuteczność Pam dla K=5:",diag5)

## Skuteczność Pam dla K=5: 0.3293173

com6<-table(etykietki.pam6,etykietki.veh)
diag6<-compareMatchedClasses(etykietki.pam6,etykietki.veh,method = "rowmax")$diag
cat("Skuteczność Pam dla K=6:",diag6)

## Skuteczność Pam dla K=6: 0.2785235
```

2.3 Zastosowanie algorytmu hierarchicznego AGNES

Zastosujemy teraz algorytm AGNES z metod hierarchicznych. Na początku zwizualizujemy dendrogramy dla trzech metod algorytmu AGNES: "average linkage", "single linkage" oraz "complete linkage" i wybierzemy jedną.

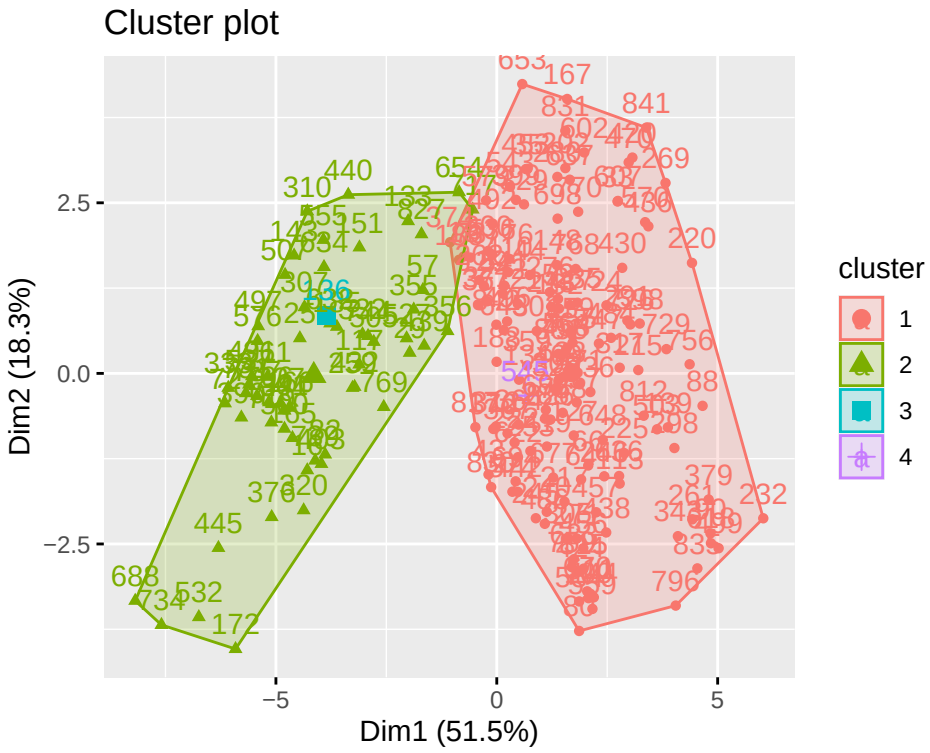
- Average linkage

```
library(cluster)
Vehicle.agnes.avg <- agnes(Vehicle3, metric="euclidean", method="average", stand=TR
Vehicle.agnes.k4 <- cutree(Vehicle.agnes.avg, k=4)
fviz_dend(Vehicle.agnes.avg, k=4, main="Dendrogram - average linkage")
```



Rysunek 11: Average linkage

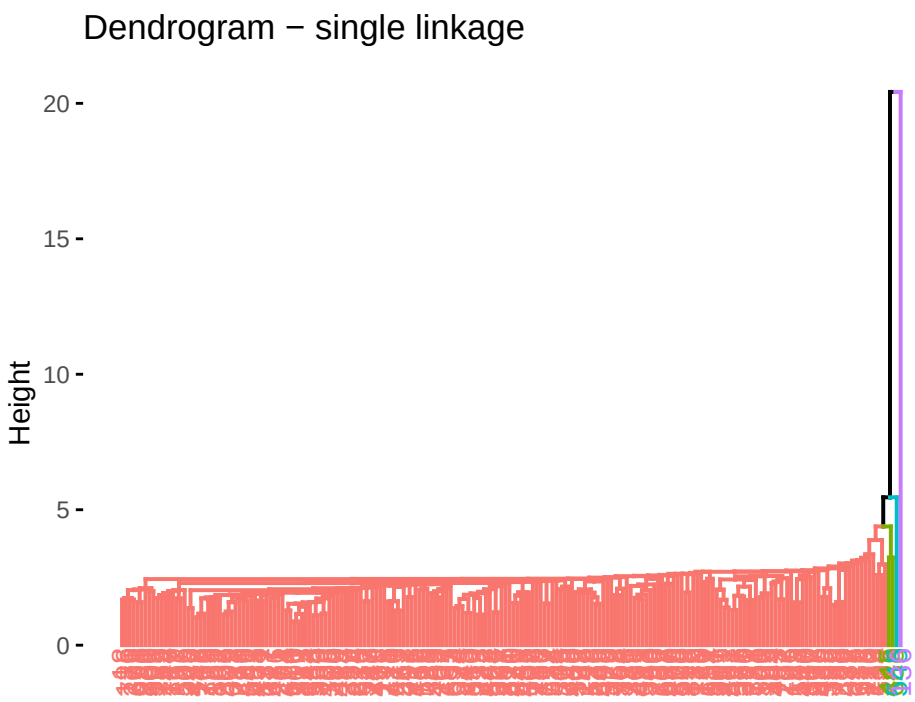
```
fviz_cluster(list(data=Vehicle3, cluster=Vehicle.agnes.k4))
```



Rysunek 12: Average linkage

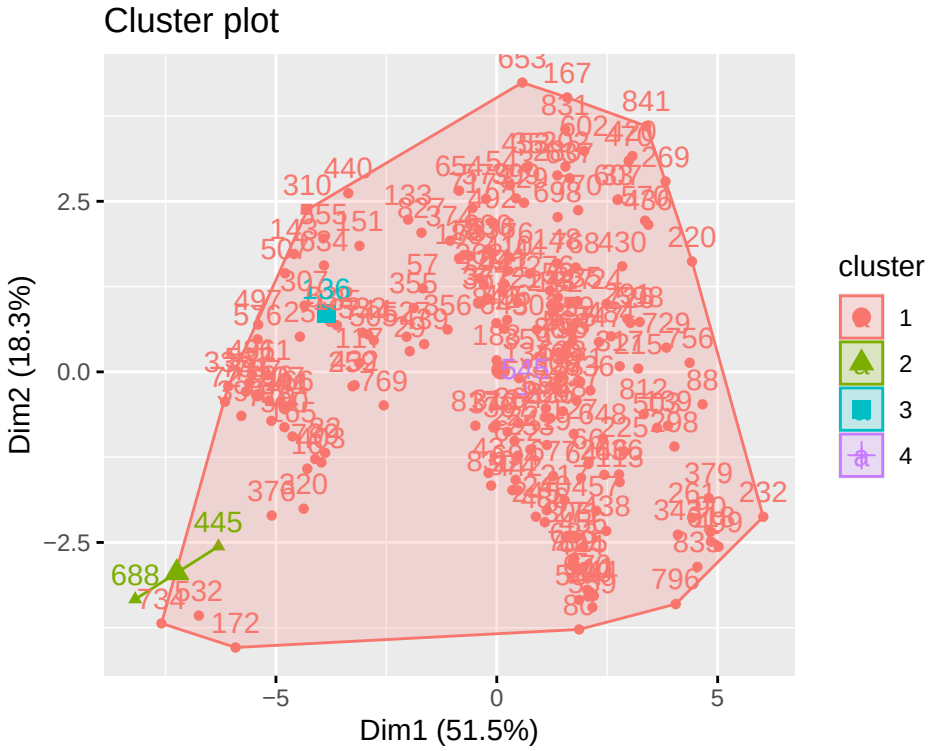
- Single linkage

```
Vehicle.agnes.sin <- agnes(Vehicle3, metric="euclidean", method="single", stand=TRU
Vehicle.agnes.k4.sin <- cutree(Vehicle.agnes.sin, k=4)
fviz_dend(Vehicle.agnes.sin, k=4, main="Dendrogram - single linkage")
```



Rysunek 13: Single linkage

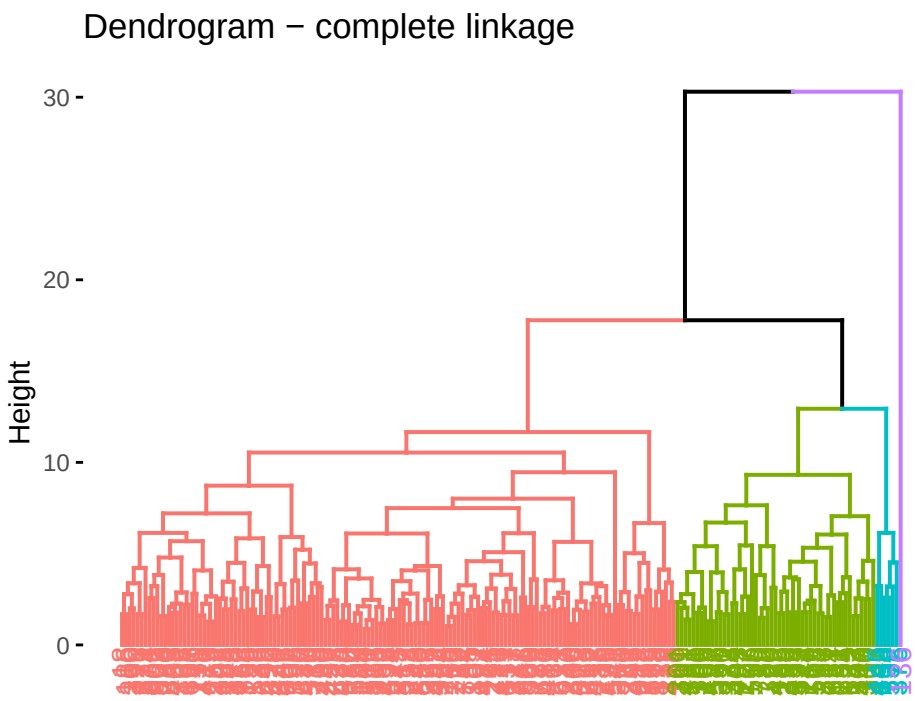
```
fviz_cluster(list(data=Vehicle3, cluster=Vehicle.agnes.k4.sin))
```



Rysunek 14: Single linkage

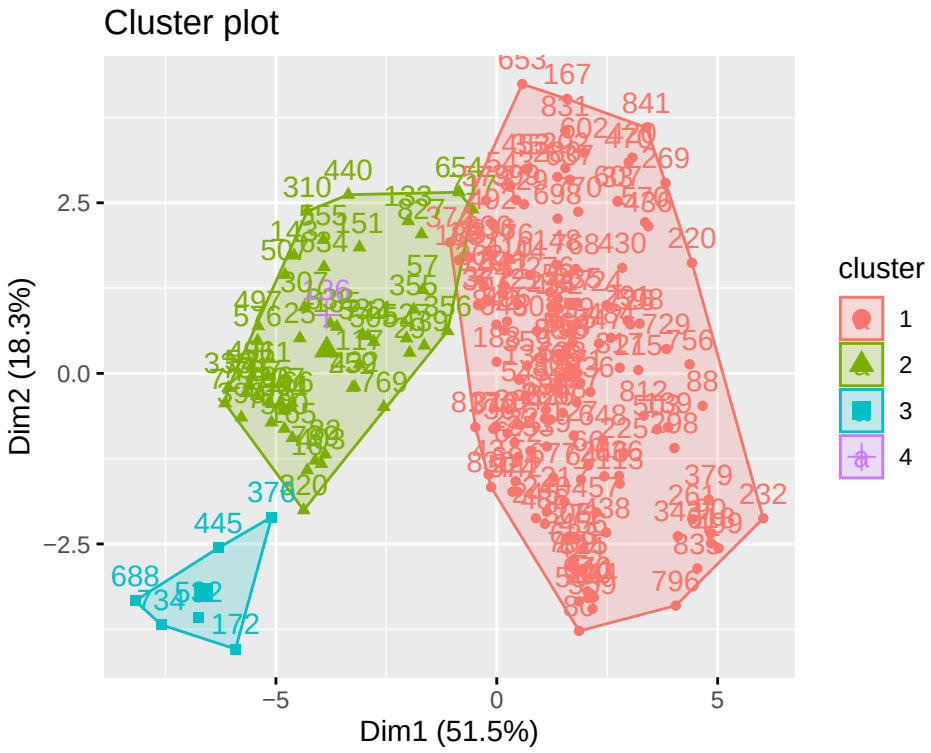
- Complete linkage

```
Vehicle.agnes.com<- agnes(Vehicle3, metric="euclidean", method="complete", stand=TR
Vehicle.agnes.k4.com <- cutree(Vehicle.agnes.com, k=4)
fviz_dend(Vehicle.agnes.com, k=4, main="Dendrogram - complete linkage")
```



Rysunek 15: Complete linkage

```
fviz_cluster(list(data=Vehicle3, cluster=Vehicle.agnes.k4.com))
```



Rysunek 16: Complete linkage

Z zaobserwowanych dendrogramów możemy stwierdzić, że najlepszym wyborem do analizy będzie metoda "complete".

2.3.1 Wybór optymalnej liczby skupień K

Z porównania współczynników silhouette wygląda na to, że wartość K=2 jest najbardziej optymalna dla naszych danych w algorytmie AGNES. Porównajmy teraz wyniki analizy do rzeczywistych wynikóW.

```
agn.diag2<-compareMatchedClasses(Vehicle.agnes.k2.com, etykietki.veh,method = "rowmax")$
cat("Zgodność z rzeczywistą przynależnością klas dla K=2 wynosi:", agn.diag2)

## Zgodność z rzeczywistą przynależnością klas dla K=2 wynosi: 0.5428571

agn.diag3<-compareMatchedClasses(Vehicle.agnes.k3.com, etykietki.veh,method = "rowmax")$
cat("Zgodność z rzeczywistą przynależnością klas dla K=3 wynosi:", agn.diag3)

## Zgodność z rzeczywistą przynależnością klas dla K=3 wynosi: 0.4965986

agn.diag4<-compareMatchedClasses(Vehicle.agnes.k4.com, etykietki.veh,method = "rowmax")$
cat("Zgodność z rzeczywistą przynależnością klas dla K=4 wynosi:", agn.diag4)

## Zgodność z rzeczywistą przynależnością klas dla K=4 wynosi: 0.3891626

agn.diag5<-compareMatchedClasses(Vehicle.agnes.k5.com, etykietki.veh,method = "rowmax")$
cat("Zgodność z rzeczywistą przynależnością klas dla K=5 wynosi:", agn.diag5)
```

```
Vehicle.agnes.k2.com <- cutree(Vehicle.agnes.com, k=2)
Vehicle.agnes.k3.com <- cutree(Vehicle.agnes.com, k=3)
Vehicle.agnes.k5.com <- cutree(Vehicle.agnes.com, k=5)
Vehicle.agnes.k6.com <- cutree(Vehicle.agnes.com, k=6)

sil.k2 <- silhouette(x=Vehicle.agnes.k2.com, dist=vehicles.macniepod.matrix)
cat("wartość średnia silhouette dla K=2:",summary(sil.k2)$avg.width)

## wartość średnia silhouette dla K=2: 0.6741887

sil.k3 <- silhouette(x=Vehicle.agnes.k3.com, dist=vehicles.macniepod.matrix)
cat("wartość średnia silhouette dla K=3:",summary(sil.k3)$avg.width)

## wartość średnia silhouette dla K=3: 0.3862943

sil.k4 <- silhouette(x=Vehicle.agnes.k4.com, dist=vehicles.macniepod.matrix)
cat("wartość średnia silhouette dla K=4:",summary(sil.k4)$avg.width)

## wartość średnia silhouette dla K=4: 0.3417172

sil.k5 <- silhouette(x=Vehicle.agnes.k5.com, dist=vehicles.macniepod.matrix)
cat("wartość średnia silhouette dla K=5:",summary(sil.k5)$avg.width)

## wartość średnia silhouette dla K=5: 0.2123498

sil.k6 <- silhouette(x=Vehicle.agnes.k6.com, dist=vehicles.macniepod.matrix)
cat("wartość średnia silhouette dla K=6:",summary(sil.k6)$avg.width)

## wartość średnia silhouette dla K=6: 0.1916359
```

```
## Zgodność z rzeczywistą przynależnością klas dla K=5 wynosi: 0.3242188

agn.diag6<-compareMatchedClasses(Vehicle.agnes.k6.com, etykietki.veh,method = "rowmax")$
cat("Zgodność z rzeczywistą przynależnością klas dla K=6 wynosi:", agn.diag6)

## Zgodność z rzeczywistą przynależnością klas dla K=6 wynosi: 0.2885246
```

Widzimy, że największą zgodność mamy dla K=2.

2.4 Podsumowanie

2.4.1 Medoidy PAM

Spójrzmy na rekordy, które są centrami skupisk w algorytmie PAM

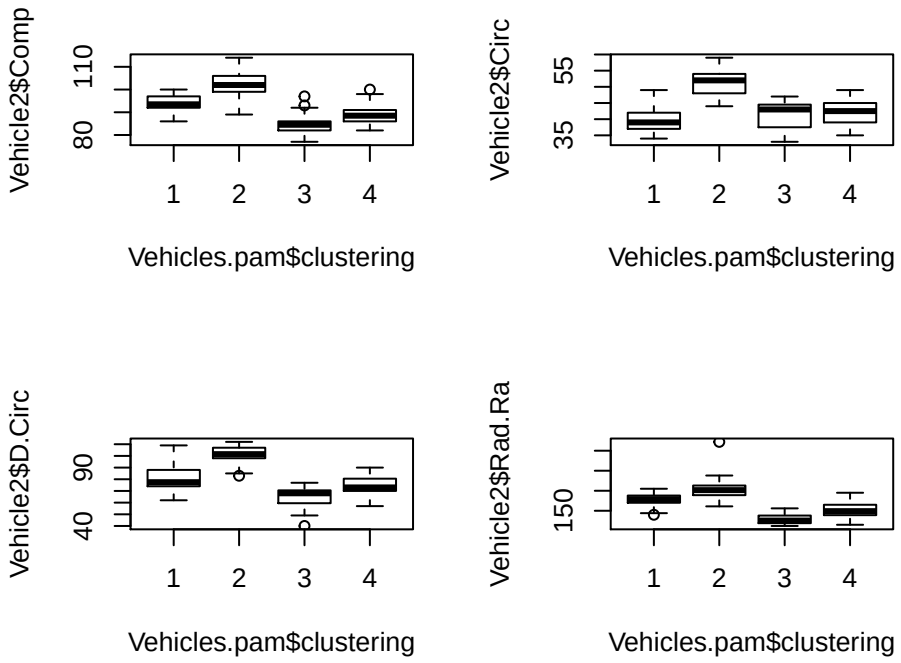
```
CentraSkupisk.nazwy <- Vehicles.pam$medoids
print(xtable(veh1[CentraSkupisk.nazwy,], table.placement='H'))
```

	Comp	Circ	D.Circ	Rad.Ra	Pr.Axis.Ra	Max.L.Ra	Scat.Ra	Elong	Pr.Axis.Rect	Max.
543	0.04	-0.86	0.37	0.47	-0.29	0.25	0.06	-0.36	-0.12	
301	1.75	1.55	1.17	0.87	-0.16	0.53	1.56	-1.43	1.49	
511	-0.88	-0.17	-0.69	-1.21	-0.69	-0.60	-0.54	0.57	-0.52	
801	-0.61	-0.51	-0.03	-0.56	0.10	-0.03	-0.60	0.44	-0.52	

2.4.2 Wykresy pudełkowe

Sprawdzimy teraz, jak wyglądają wykresy pudełkowe dla poszczególnych (losowo) wybranych cech.

```
par(mfrow=c(2,2))
boxplot(Vehicle2$Comp~Vehicles.pam$clustering)
boxplot(Vehicle2$Circ~Vehicles.pam$clustering)
boxplot(Vehicle2$D.Circ~Vehicles.pam$clustering)
boxplot(Vehicle2$Rad.Ra~Vehicles.pam$clustering)
```



Rysunek 17: Boxploty dla losowych cech

Możemy zaobserwować, że dla wybranych cech, wyniki nie są bardzo rozbieżne dla różnych klas, co świadczy o trudności przyporządkowania jednoznacznie niektórych obiektóW.

2.4.3 Wnioski końcowe

Przechodząc przez całą analizę, dochodzimy do wniosku że nasze dane są ciężkie do jednoznacznej klasyfikacji, ponieważ w najlepszych warunkach, dla optymalnego K osiągamy zgodność na poziomie około 60 procent. Metoda PAM osiąga lepsze wyniki dla naszych danych niż metoda AGNES.