

Raport 3

Eksploracja danych

Mikołaj Langner, Marcin Kostrzewa

nr albumów: 255716, 255749

2021-05-28

Spis treści

| | | |
|----------|---|----------|
| 1 | Wstęp | 1 |
| 2 | Zadanie 1 | 2 |
| 2.1 | Wczytanie danych i podział na zbiór uczący i testowy | 2 |
| 2.2 | Konstrukcja klasyfikatora i wyznaczenie prognoz | 2 |
| 2.3 | Ocena jakości klasyfikacji | 4 |
| 2.4 | Zastosowanie regresji liniowej do modelu o rozszerzonej ilości cech | 4 |
| 2.5 | Wnioski | 6 |
| 3 | Zadanie 2 | 7 |
| 3.1 | Wczytanie i krótka analiza danych | 7 |
| 3.2 | Metoda k-najbliższych sąsiadów | 8 |
| 3.3 | Drzewa klasyfikacyjne | 13 |
| 3.4 | Naiwny klasyfikator bayesowski | 17 |
| 3.5 | Wieloklasowa regresja logistyczna | 19 |
| 3.6 | Podsumowanie | 21 |

1 Wstęp

Raport zawiera rozwiązania listy 3.

W zadaniu pierwszym zbudujemy klasyfikator na bazie metody regresji liniowej i oceniamy jego skuteczność i dokładność.

W zadaniu drugim porównamy ze sobą rezultaty zastosowania następujących metod klasyfikacji:

- metoda k-najbliższych sąsiadów (*k-Nearest Neighbors*),
- drzewa klasyfikacyjne (*classification trees*),

- naiwny klasyfikator bayesowski (*naïve Bayes classifier*),
- wieloklasowa regresja logistyczna (*multinomial logistic regression*).

2 Zadanie 1

2.1 Wczytanie danych i podział na zbiór uczący i testowy

Wczytajmy dane o irysach i podzielmy je na zbiór uczący i testowy w proporcji 1 : 2.

```
data(iris)
n <- dim(iris)[1]

train.set.index <- sample(1:n, 2/3*n)
train.set <- iris %>% slice(train.set.index) %>% arrange(Species)
test.set <- iris %>% slice(-train.set.index) %>% arrange(Species)
```

2.2 Konstrukcja klasyfikatora i wyznaczenie prognoz

Stworzymy teraz macierze eksperymentu i wskaźnikową zarówno dla zbioru uczącego, jak i testowego. W tym celu wykorzystamy funkcję `dummyVars` z pakietu `caret`.

```
dummies <- dummyVars(" ~ .", data=iris)

train.dummies <- predict(dummies, newdata = train.set)
train.X <- as.matrix(cbind(rep(1, nrow(train.dummies)),
                           train.dummies[, 1:4]))
train.Y <- train.dummies[, 5:7]

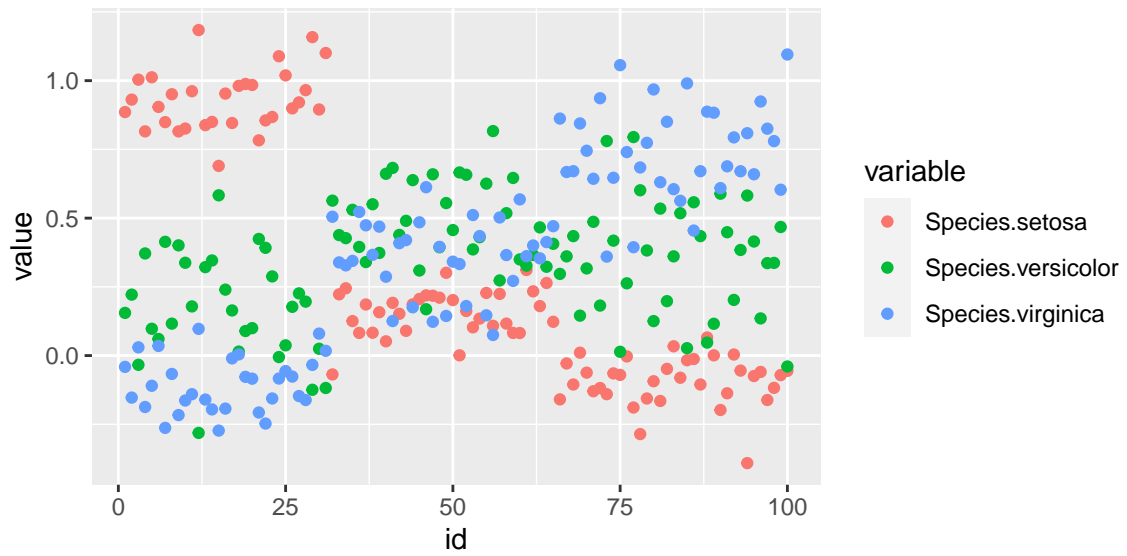
test.dummies <- predict(dummies, newdata = test.set)
test.X <- as.matrix(cbind(rep(1, nrow(test.dummies)), test.dummies[, 1:4]))
test.Y <- test.dummies[, 5:7]
```

Wykorzystując metodę najmniejszych kwadratów, wyznaczamy przewidywane prognozy klas dla obu zbiorów.

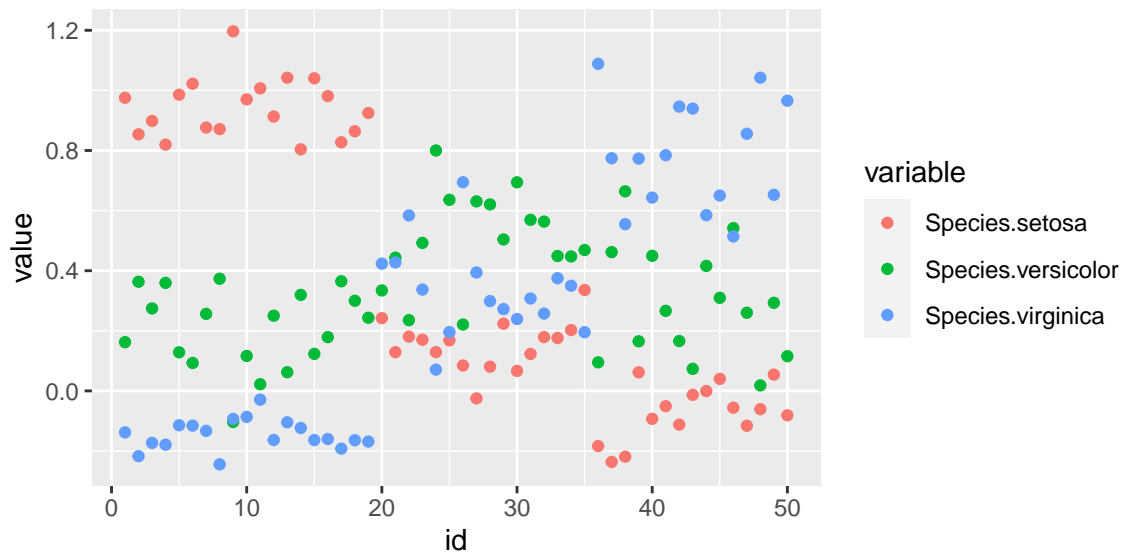
```
Y.hat <- solve(t(train.X) %*% train.X) %*% t(train.X) %*% train.Y

train.proba <- train.X %*% Y.hat
test.proba <- test.X %*% Y.hat
```

Przedstawmy prognozy klas na wykresach.



Rysunek 1: Prognozy klas dla zbioru uczącego.



Rysunek 2: Prognozy klas dla zbioru testowego.

2.3 Ocena jakości klasyfikacji

Wyznaczymy teraz macierz pomyłek dla zbioru uczącego.

| | Species.setosa | Species.versicolor | Species.virginica |
|------------|----------------|--------------------|-------------------|
| setosa | 31 | 0 | 0 |
| versicolor | 0 | 20 | 14 |
| virginica | 0 | 3 | 32 |

Tabela 1: Macierz pomyłek dla zbioru uczącego.

Błąd klasyfikacji to 0.17.

Podobnie, wyznaczmy teraz macierz pomyłek dla zbioru testowego.

| | Species.setosa | Species.versicolor | Species.virginica |
|------------|----------------|--------------------|-------------------|
| setosa | 19 | 0 | 0 |
| versicolor | 0 | 13 | 3 |
| virginica | 0 | 2 | 13 |

Tabela 2: Macierz pomyłek dla zbioru testowego.

Błąd klasyfikacji wynosi 0.1.

Przypatując się wykresom (1), (2) możemy zauważyć, że zachodzi zjawisko maskowania — klasa `versicolor` jest przysłaniana.

2.4 Zastosowanie regresji liniowej do modelu o rozszerzonej ilości cech

Najpierw uzupełnijmy dane o irysach o składniki wielomianowe stopnia 2.

```
iris.quad <- (iris %>% select(-Species))^2
colnames(iris.quad) <- c("SL^2", "SW^2", "PL^2", "PW^2")
iris <- cbind(iris, combn(iris %>% select(-Species), 2,
                        FUN = Reduce, f = `*`),
            iris.quad)
```

Podobnie jak poprzednio podzielimy dane na zbiory: uczący i testowy, a następnie utworzymy macierze: eksperymentu i indyktorów.

```
train.set.index <- sample(1:n, 2/3*n)
train.set <- iris %>% slice(train.set.index) %>% arrange(Species)
test.set <- iris %>% slice(-train.set.index) %>% arrange(Species)

dummies <- dummyVars(" ~ .", data=iris)
```

```

train.dummies <- predict(dummies, newdata = train.set)
train.X <- as.matrix(cbind(rep(1, nrow(train.dummies)), train.dummies[, -c(5:7)]))
train.Y <- train.dummies[, 5:7]
test.dummies <- predict(dummies, newdata = test.set)
test.X <- as.matrix(cbind(rep(1, nrow(test.dummies)), test.dummies[, -c(5:7)]))
test.Y <- test.dummies[, 5:7]

```

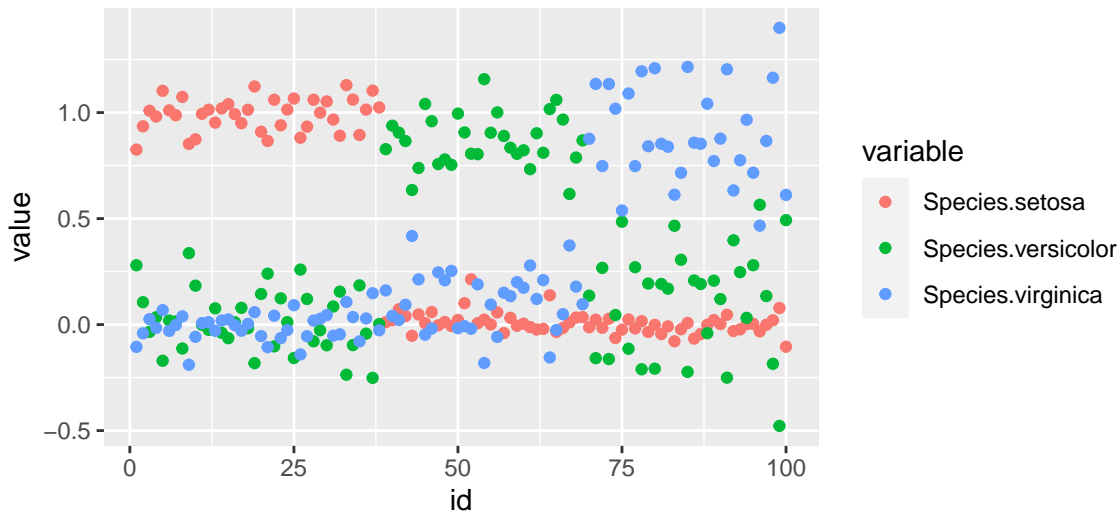
Ponownie, wyznaczmy prognozy klas i zwizualizujemy to przypisanie na wykresach.

```

Y.hat <- solve(t(train.X) %*% train.X) %*% t(train.X) %*% train.Y

train.proba <- train.X %*% Y.hat
test.proba <- test.X %*% Y.hat

```



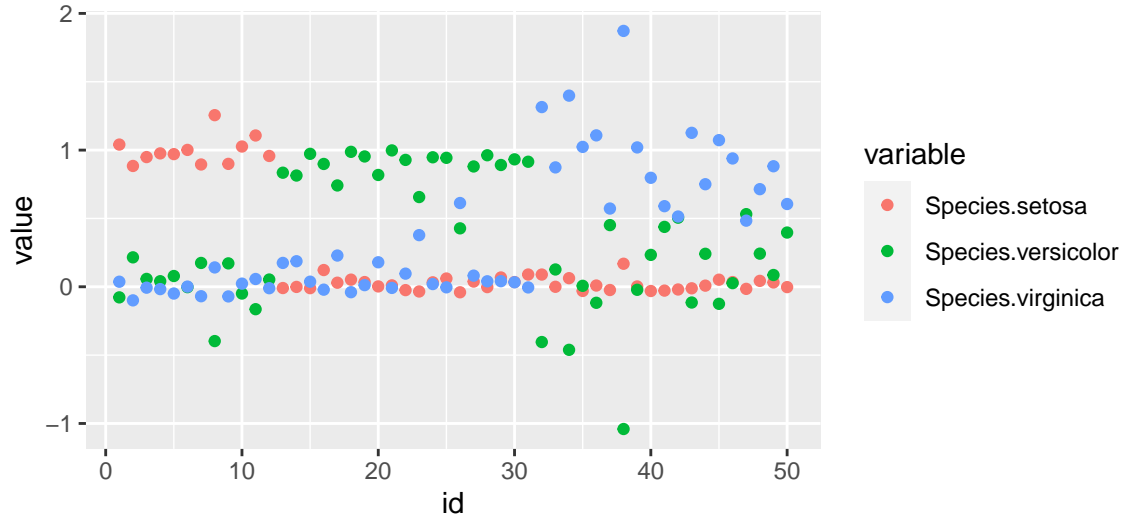
Rysunek 3: Prognozy klas dla zbioru uczacego o rozszerzonej liczbie cech.

Wyznaczymy także macierze pomyłek i błędy klasyfikacji.

| | Species.setosa | Species.versicolor | Species.virginica |
|------------|----------------|--------------------|-------------------|
| setosa | 38 | 0 | 0 |
| versicolor | 0 | 31 | 0 |
| virginica | 0 | 1 | 30 |

Tabela 3: Macierz pomyłek dla zbioru uczacego dla przypadku o rozszerzonej liczbie cech.

Błąd klasyfikacji wynosi 0.01.



Rysunek 4: Prognozy klas dla zbioru uczącego o rozszerzonej liczbie cech.

| | Species.setosa | Species.versicolor | Species.virginica |
|------------|----------------|--------------------|-------------------|
| setosa | 12 | 0 | 0 |
| versicolor | 0 | 18 | 1 |
| virginica | 0 | 1 | 18 |

Tabela 4: Macierz pomyłek dla zbioru testowego dla przypadku o rozszerzonej liczbie cech.

Błąd klasyfikacji wynosi 0.04.

Po przeanalizowaniu wykresów (3) i (4) dochodzimy do wniosku, że w tym przypadku zjawisko maskowania klas zostało zniwelowane.

2.5 Wnioski

Przede wszystkim zauważamy, że model oparty na rozszerzonej ilości cech dał znacznie lepsze rezultaty — błędy klasyfikacji były mniejsze zarówno dla zbioru uczącego, jak i testowego. W drugim modelu nie wystąpiło także zjawisko maskowania.

3 Zadanie 2

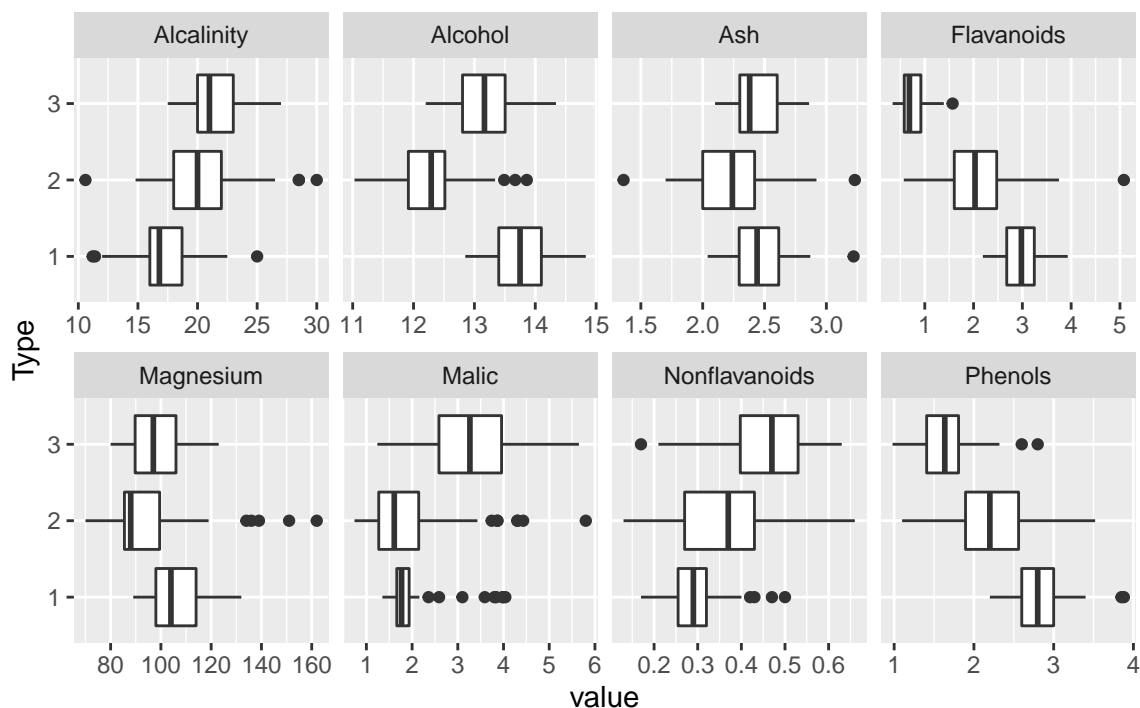
3.1 Wczytanie i krótka analiza danych

Wczytajmy i zapoznajmy się z danymi.

```
data(wine)
n <- dim(wine)[1]
variable.number <- ncol(wine)
observations.number <- nrow(wine)
NaN.number <- sum(is.na(wine))
class.number <- length(unique(wine$Type))
```

Mamy 14 zmiennych i 178 obserwacji. Są trzy klasy, informację o nich zawiera zmienna `Type`. Nie występują wartości brakujące (`NaN.number = 0`).

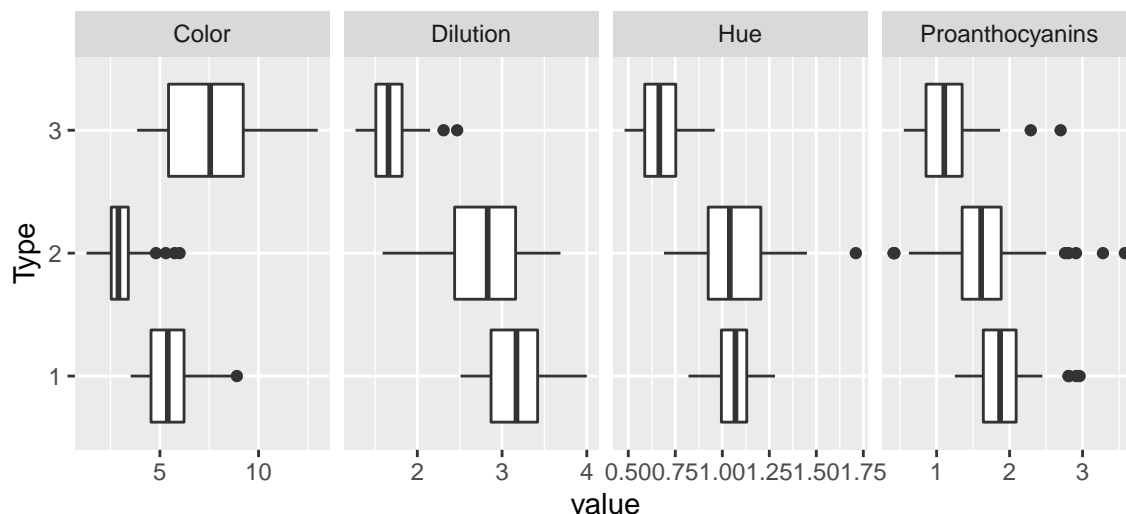
Przyjrzyjmy się naszym danym na wykresach pudełkowych — wykresy (5) i (6).



Rysunek 5: Wykresy pudełkowe naszych danych.

Możemy zauważyć, że zmiennymi, które dobrze dywersyfikują klasy są: `Alcohol`, `Flavanoids` i `Phenols`.

Podzielimy nasze dane na zbiór uczący i testowy w stosunku 2 : 1. Utworzymy także podzbiory, które będą zawierać tylko najlepiej dywersyfikujące cechy.



Rysunek 6: Wykresy pudełkowe naszych danych.

```
set.seed(42)
train.index <- sample(n, 2/3 * n)
train.data <- wine %>% slice(train.index)
test.data <- wine %>% slice(-train.index)
train.subset <- data.frame(train.data[, c(1, 2, 7, 8)])
test.subset <- data.frame(test.data[, c(1, 2, 7, 8)])
```

Zdefiniujmy też od razu rzeczywiste etykiety klas dla wcześniej utworzonych zbiorów.

```
train.etiquettes <- train.data$Type
test.etiquettes <- test.data$Type
subset.train.etiquettes <- train.subset$Type
subset.test.etiquettes <- test.subset$Type
```

Poniżej tworzymy także obiekt `trainControl`, który wykorzystamy przy przeprowadzaniu 5-krotnej walidacji krzyżowej.

```
cv <- trainControl(method="cv", number=5)
```

3.2 Metoda k-najbliższych sąsiadów

Na początku wytrenujemy nasz klasyfikator na zbiorze uczącym zawierającym wszystkie cechy (przyjmujemy $k = 5$).

```
model.knn.basic <- ipredknn(Type ~ ., data = train.data, k=5)

basic.knn.test.pred <- predict(model.knn.basic, test.data, type="class")
basic.knn.train.pred <- predict(model.knn.basic, train.data, type="class")
```


Wyznaczmy dla niego macierze pomyłek.

| | 1 | 2 | 3 |
|---|----|----|----|
| 1 | 32 | 5 | 1 |
| 2 | 2 | 39 | 8 |
| 3 | 2 | 7 | 22 |

(a) Zbior uczacy

| | 1 | 2 | 3 |
|---|----|----|---|
| 1 | 21 | 1 | 2 |
| 2 | 0 | 15 | 9 |
| 3 | 2 | 4 | 6 |

(b) Zbior testowy

Tabela 5: Macierze pomyłek dla metody KNN — wszystkie cechy.

Błędy klasyfikacji to 0.2118644 i 0.3.

Teraz stworzymy klasyfikator na zbiorze uczącym zawierającym wybrane cechy (przyjmujemy $k = 5$).

```
knn.model.subset <- ipredknn(Type ~ ., data = train.subset, k=5)

subset.knn.test.pred <- predict(knn.model.subset, test.subset, type="class")
subset.knn.train.pred <- predict(knn.model.subset, train.subset, type="class")
```

Macierze pomyłek wyglądają następująco:

| | 1 | 2 | 3 |
|---|----|----|----|
| 1 | 35 | 3 | 0 |
| 2 | 1 | 46 | 0 |
| 3 | 0 | 2 | 31 |

(a) Zbior uczacy

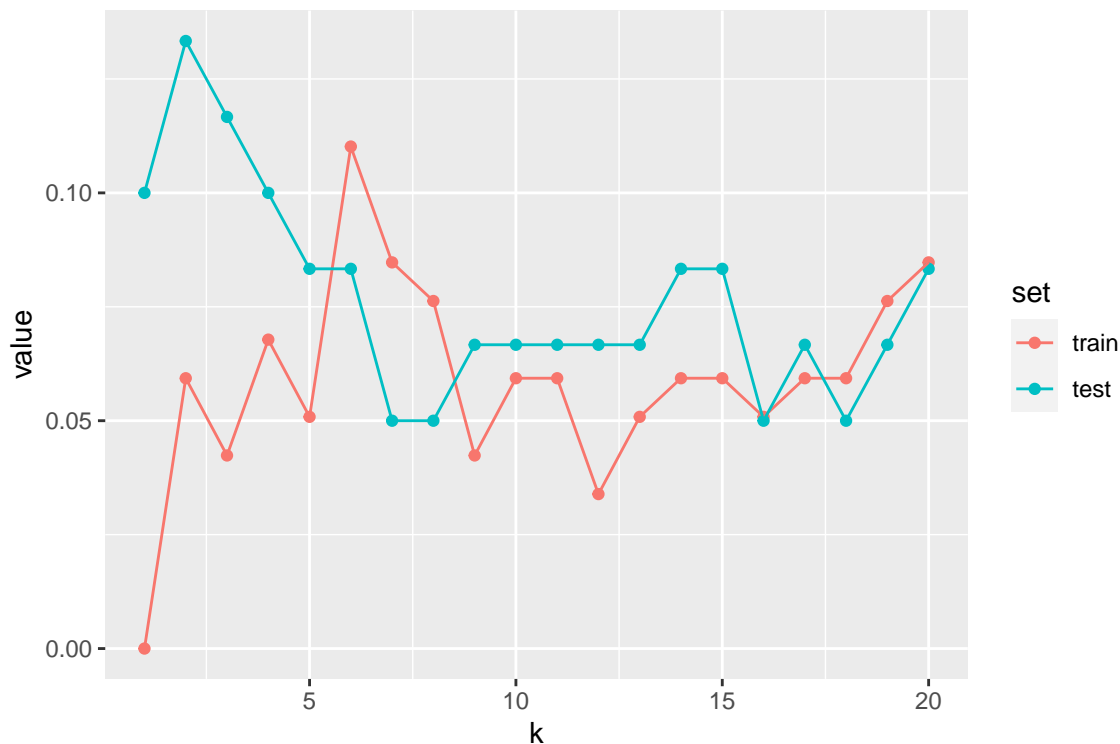
| | 1 | 2 | 3 |
|---|----|----|----|
| 1 | 22 | 2 | 0 |
| 2 | 1 | 16 | 0 |
| 3 | 0 | 2 | 17 |

(b) Zbior testowy

Tabela 6: Macierze pomyłek dla metody KNN — wybrane cechy.

Błędy klasyfikacji to 0.0508475 i 0.0833333.

Widzimy, że klasyfikator wyćwiczony na `train.subset` poradził sobie lepiej. Zobaczmy więc jak zmienia się błąd klasyfikacji w zależności od parametru k na zbiorze z ograniczoną liczbą cech.



Rysunek 7: Błąd klasyfikacji w zależności od parametru k .

Widzimy, że najlepsze rezultaty otrzymujemy dla $k = 9$.

Wykorzystamy teraz pakiet `caret` do stworzenia modelu stuningowanego. By taki model powstał, wykorzystamy 5-krotną walidację krzyżową.

```
model <- train(Type ~ ., data = train.subset, method = "knn", trControl = cv)

tuned.knn.test.pred <- predict(model, test.data)
tuned.knn.train.pred <- predict(model, train.data)
```

Jak się okazuje, model ten również przyjmuje $k = 9$.

Dla tego klasyfikatora także wyznaczmy macierze pomyłek i błędy klasyfikacji.

| | 1 | 2 | 3 | | 1 | 2 | 3 |
|------------------|----|----|----|-------------------|----|----|----|
| 1 | 35 | 2 | 0 | 1 | 23 | 2 | 0 |
| 2 | 1 | 47 | 0 | 2 | 0 | 18 | 2 |
| 3 | 0 | 2 | 31 | 3 | 0 | 0 | 15 |
| (a) Zbior uczący | | | | (b) Zbior testowy | | | |

Tabela 7: Macierze pomyłek dla metody KNN — stuningowany model.

Błędy klasyfikacji to 0.0423729 i 0.0666667.

Jak widzimy model stuningowany poradził sobie najlepiej. Wyznamy teraz dla niego błąd predykcji — skorzystamy z 5-krotnej walidacji krzyżowej, metody bootstrap oraz .632+.

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

knn.predictor <- function(formula, data)
{ train(formula, data = data, method = "knn", trControl = cv)}

knn.error.cv <- errorest(Type~., wine[, c(1, 2, 7, 8)], model=knn.predictor,
                        predict=predictor, estimator="cv",
                        est.para=control.errorest(k = 5))

knn.error.boot <- errorest(Type~., wine[, c(1, 2, 7, 8)], model=knn.predictor,
                        predict=predictor, estimator="boot",
                        est.para=control.errorest(nboot = 25))

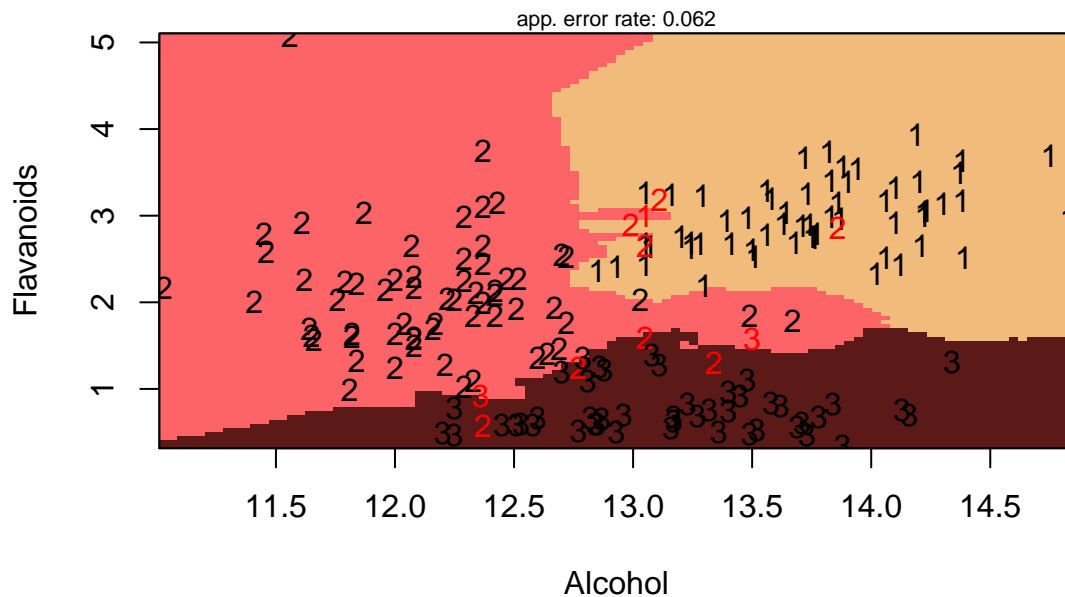
knn.error.632 <- errorest(Type~., wine[, c(1, 2, 7, 8)], model=knn.predictor,
                        predict=predictor, estimator="632plus",
                        est.para=control.errorest(nboot = 25))
```

Błędy wyniosły kolejno 0.0842697, 0.1011415 oraz 0.0822332.

```
library(klaR)

## Warning: package 'klaR' was built under R version 4.0.5
## Loading required package: MASS
##
## Attaching package: 'MASS'
## The following object is masked from 'package:dplyr':
##
##      select

drawparti(wine$Type, x=wine$Alcohol, y=wine$Flavanoids, method = "sknn",
          plot.matrix = FALSE, imageplot = TRUE, col.contour = "red",
          col.wrong = "red", image.colors = wes_palette("GrandBudapest1", 3),
          col.correct = "black", xlab="Alcohol", ylab="Flavanoids")
```



```

model <- train(Type ~ Alcohol + Flavanoids, data = train.subset, method = "knn", trCont

tuned.knn.test.pred <- predict(model, test.data)
tuned.knn.train.pred <- predict(model, train.data)

tuned.predictions <- predict(model, test.subset)

var1 <- wine$Alcohol
var2 <- wine$Flavanoids

var3 <- test.subset$Alcohol
var4 <- test.subset$Flavanoids

tuned.predictions <- predict(model, test.subset)

test.confusion <- table(tuned.predictions, test.etiquettes)

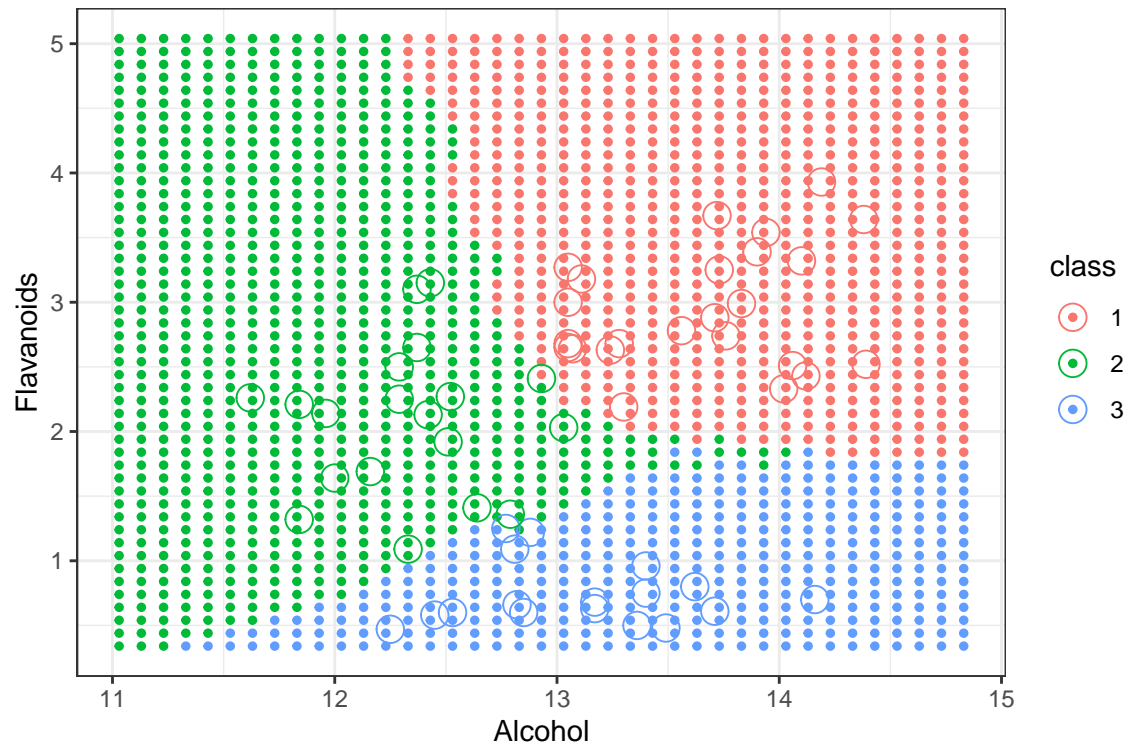
x.test <- expand.grid(Alcohol = seq(min(var1), max(var1), by = 0.1),
                     Flavanoids = seq(min(var2), max(var2), by=0.1))

x.test.pred <- predict(model, x.test)

df1 <- data.frame(x.test, class = x.test.pred)
df2 <- data.frame(x = var3, y = var4, class = tuned.predictions)

```

```
ggplot() +
  geom_point(aes(x=Alcohol, y=Flavanoids, col = class), data = df1, size = 1) +
  geom_point(aes(x = x, y = y, col = class), data = df2, size = 4.5, shape = 1) +
  theme_bw()
```



3.3 Drzewa klasyfikacyjne

Najpierw wytrenujemy model na zbiorze uczącym zawierającym wszystkie cechy.

```
basic.tree.model <- rpart(Type ~ ., data = train.data)

basic.tree.test.pred <- predict(basic.tree.model, newdata = test.data,
                                type = "class")
basic.tree.train.pred <- predict(basic.tree.model, newdata = train.data,
                                type = "class")
```

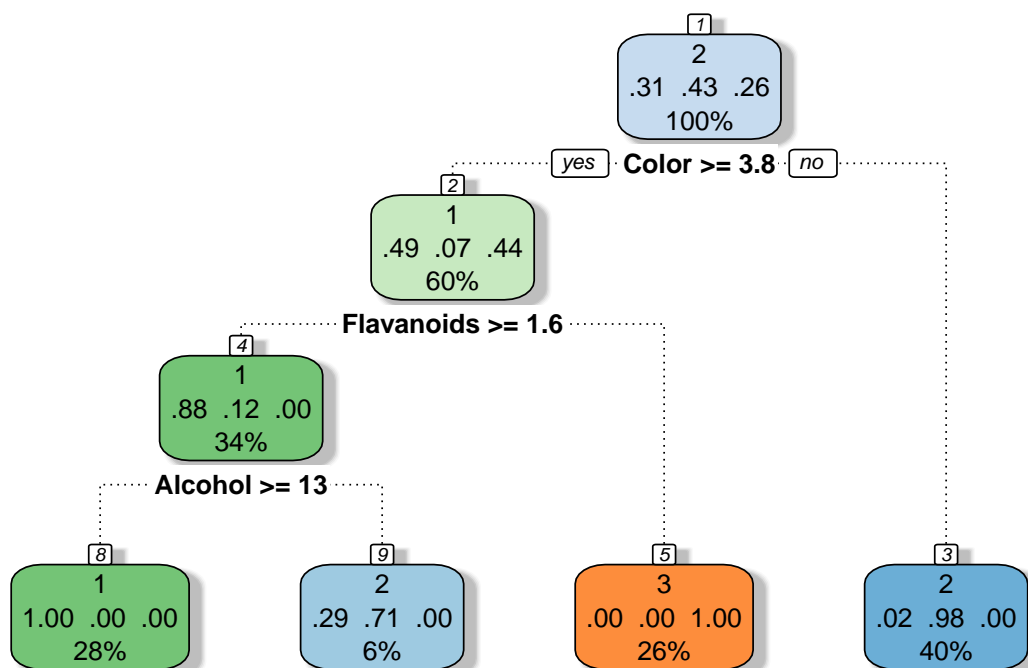
To drzewo klasyfikacyjne wygląda następująco (8).

Wyznamy dla tego modelu macierze pomyłek i błędy klasyfikacji.

Błędy klasyfikacji to 0.0254237 i 0.1333333.

Teraz stworzymy model, który wytrenujemy na `train.subset`.

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



Rysunek 8: Drzewo decyzyjne — wszystkie cechy.

| | 1 | 2 | 3 | | 1 | 2 | 3 |
|------------------|----|----|----|-------------------|----|----|----|
| 1 | 33 | 0 | 0 | 1 | 17 | 1 | 0 |
| 2 | 3 | 51 | 0 | 2 | 6 | 18 | 0 |
| 3 | 0 | 0 | 31 | 3 | 0 | 1 | 17 |
| (a) Zbiór uczący | | | | (b) Zbiór testowy | | | |

Tabela 8: Macierze pomyłek dla metody drzew klasyfikacyjnych — wszystkie cechy.

```

subset.tree.test.pred <- predict(subset.tree.model, newdata = test.subset,
                                type = "class")
subset.tree.train.pred <- predict(subset.tree.model, newdata = train.subset,
                                type = "class")

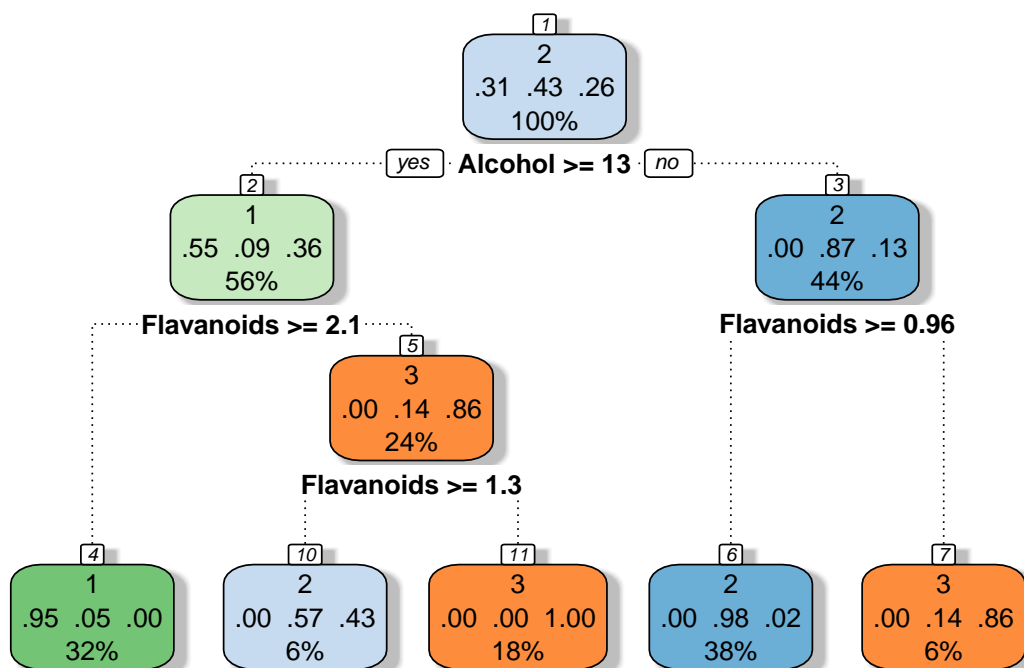
```

To drzewo decyzyjne wygląda następująco (9).

Podobnie jak wcześniej, wyznaczmy dla niego macierze pomyłek i błędy klasyfikacji.

Błędy klasyfikacji to 0.059322 i 0.0666667.

Ponownie stworzymy stuningowany model, zmieniając parametr `cp`, przy pomocy 5-krotnej walidacji krzyżowej.



Rysunek 9: Drzewo decyzyjne — wybrane cechy.

| | 1 | 2 | 3 | | 1 | 2 | 3 |
|------------------|----|----|----|-------------------|----|----|----|
| 1 | 36 | 2 | 0 | 1 | 23 | 2 | 0 |
| 2 | 0 | 48 | 4 | 2 | 0 | 17 | 1 |
| 3 | 0 | 1 | 27 | 3 | 0 | 1 | 16 |
| (a) Zbiór uczący | | | | (b) Zbiór testowy | | | |

Tabela 9: Macierze pomyłek dla metody drzew klasyfikacyjnych — wybrane cechy.

```

tuned.tree.model <- train(Type ~ ., data = train.subset, method = "rpart",
                           trControl = cv, tuneLength = 10)

tuned.tree.test.pred <- predict(tuned.tree.model, test.subset)
tuned.tree.train.pred <- predict(tuned.tree.model, train.subset)

```

Błędy klasyfikacji to 0.059322 i 0.0666667.

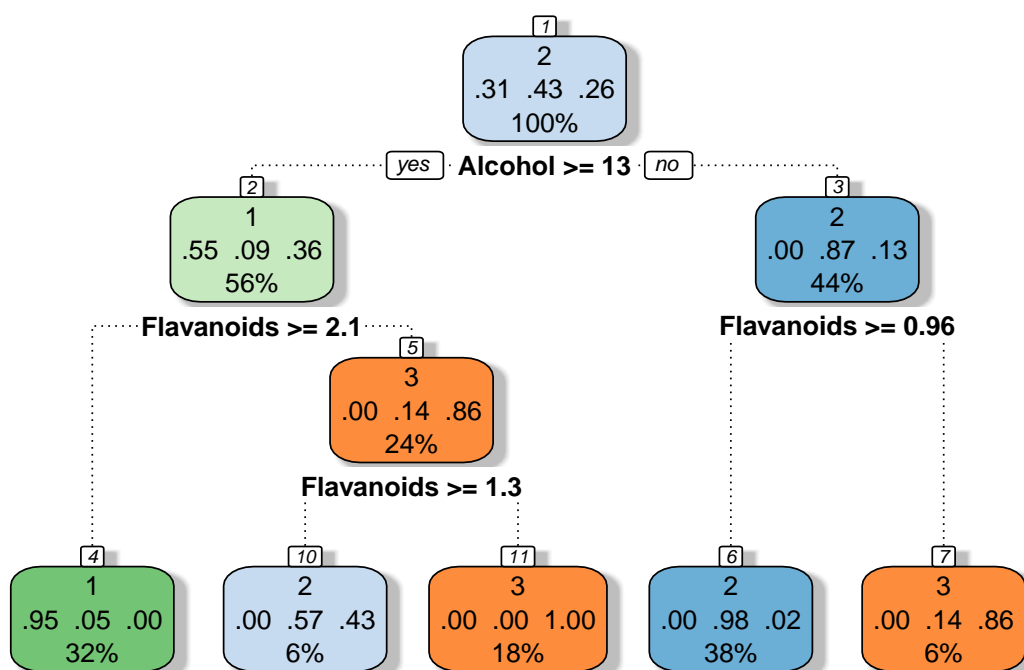
Zobaczmy jak wpłynie na model wytrenowany na `train.subset` zmiana parametrów drzewa — tabela (11).

```

search_grid <- expand.grid(cp=seq(0,0.1,0.001))

best.model.tree <- train(Type ~., data = train.subset, method = "rpart",

```



Rysunek 10: Drzewo decyzyjne — stuningowany model.

| | 1 | 2 | 3 | | 1 | 2 | 3 |
|---|----|----|----|---|----|----|----|
| 1 | 36 | 2 | 0 | 1 | 23 | 2 | 0 |
| 2 | 0 | 48 | 4 | 2 | 0 | 17 | 1 |
| 3 | 0 | 1 | 27 | 3 | 0 | 1 | 16 |

(a) Zbior uczacy (b) Zbior testowy

Tabela 10: Macierze pomyłek dla metody drzew klasyfikacyjnych — stuningowany model.

```

trControl = cv, tuneGrid = search_grid)

a <- best.model.tree$results %>% top_n(5, wt = Accuracy) %>%
  arrange(desc(Accuracy))

```

| | cp | Accuracy | Kappa | AccuracySD | KappaSD |
|---|-------|----------|-------|------------|---------|
| 1 | 0.000 | 0.859 | 0.789 | 0.147 | 0.218 |
| 2 | 0.001 | 0.859 | 0.789 | 0.147 | 0.218 |
| 3 | 0.002 | 0.859 | 0.789 | 0.147 | 0.218 |
| 4 | 0.003 | 0.859 | 0.789 | 0.147 | 0.218 |
| 5 | 0.004 | 0.859 | 0.789 | 0.147 | 0.218 |

Tabela 11: 5 najlepszych modeli drzewa decyzyjnego.

Widzimy, że najlepsze rezultaty otrzymujemy dla $cp = 0$ — drzewa nie trzeba w ogóle przycinać. Jak widzimy model stuningowany poradził sobie najlepiej. Wyznamy teraz dla niego błąd predykcji — skorzystamy ponownie z 5-krotnej walidacji krzyżowej, metody bootstrap oraz .632+.

```
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, 7, 8)],
                                model=decision.tree.predictor,
                                predict=predictor, estimator="cv",
                                est.param=control.errorest(k = 5))

decision.tree.error.boot <- errorest(Type~., wine[, c(1, 2, 7, 8)],
                                model=decision.tree.predictor,
                                predict=predictor, estimator="boot",
                                est.param=control.errorest(nboot = 25))

decision.tree.error.632 <- errorest(Type~., wine[, c(1, 2, 7, 8)],
                                model=decision.tree.predictor,
                                predict=predictor, estimator="632plus",
                                est.param=control.errorest(nboot = 25))
```

Błędy wyniosły kolejno 0.1067416, 0.1119916 oraz 0.0940631.

3.4 Naiwny klasyfikator bayesowski

Najpierw wytrenujemy model na zbiorze uczącym zawierającym wszystkie zmienne.

```
bayes.model.basic <- naiveBayes(Type ~ ., data = train.data)

basic.bayes.train.pred <- predict(bayes.model.basic, train.data)
basic.bayes.test.pred <- predict(bayes.model.basic, test.data)
```

Wyznamy dla tego modelu macierze pomyłek i wartości błędów klasyfikacji.

Błędy klasyfikacji to kolejno 0.0084746 i 0.0333333.

Powtórzmy teraz powyższe dla wybranego podzbioru naszych danych.

```
bayes.model.subset <- naiveBayes(Type ~ ., data = train.subset)

bayes.subset.train.pred <- predict(bayes.model.subset, train.subset)
bayes.subset.test.pred <- predict(bayes.model.subset, test.subset)
```

| | 1 | 2 | 3 | | 1 | 2 | 3 |
|------------------|----|----|----|-------------------|----|----|----|
| 1 | 36 | 0 | 0 | 1 | 22 | 1 | 0 |
| 2 | 0 | 50 | 1 | 2 | 0 | 19 | 1 |
| 3 | 0 | 0 | 31 | 3 | 0 | 0 | 17 |
| (a) Zbior uczacy | | | | (b) Zbior testowy | | | |

Tabela 12: Macierze pomyłek dla klasyfikatora bayesowskiego — wszystkie cechy.

| | 1 | 2 | 3 | | 1 | 2 | 3 |
|------------------|----|----|----|-------------------|----|----|----|
| 1 | 34 | 2 | 0 | 1 | 20 | 3 | 0 |
| 2 | 2 | 47 | 2 | 2 | 2 | 17 | 1 |
| 3 | 0 | 1 | 30 | 3 | 0 | 0 | 17 |
| (a) Zbior uczacy | | | | (b) Zbior testowy | | | |

Tabela 13: Macierze pomyłek dla klasyfikatora bayesowskiego — wybrane cechy.

Błędy klasyfikacji to kolejno 0.059322 i 0.1.

Ponownie skorzystamy z pakietu `caret`, by stworzyć model stuningowany, wytrenowany na wszystkich cechach. Wyznamy dla niego macierze pomyłek i błędy klasyfikacji.

```
model <- train(Type ~ ., data = train.data, method = "naive_bayes",
               trControl = cv)
```

| | 1 | 2 | 3 | | 1 | 2 | 3 |
|------------------|----|----|----|-------------------|----|----|----|
| 1 | 36 | 0 | 0 | 1 | 22 | 1 | 0 |
| 2 | 0 | 50 | 1 | 2 | 0 | 19 | 1 |
| 3 | 0 | 0 | 31 | 3 | 0 | 0 | 17 |
| (a) Zbior uczacy | | | | (b) Zbior testowy | | | |

Tabela 14: Macierze pomyłek dla klasyfikatora bayesowskiego — model stuningowany.

Błędy klasyfikacji w tym przypadku to kolejno 0.0084746 i 0.0333333.

Widzimy, że poradził on sobie najlepiej z trzech rozważanych modeli.

Powtórzymy teraz ocenę klasyfikacji, podobnie jak dla wcześniej, dla stuningowanego klasyfikatora bayesowskiego.

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

naiveBayes.predictor <- function(formula, data)
{ train(formula, data = data, method = "naive_bayes", trControl = cv)}
```

```

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

naiveBayes.error.boot <- errorest(Type~., wine, model=naiveBayes.predictor,
                                predict=predictor, estimator="boot",
                                est.param=control.errorest(nboot = 25))

naiveBayes.error.632 <- errorest(Type~., wine, model=naiveBayes.predictor,
                                predict=predictor, estimator="632plus",
                                est.param=control.errorest(nboot = 25))

```

Błędy predykcji wyniosły kolejno 0.0337079, 0.0276237 oraz 0.0195704.

3.5 Wieloklasowa regresja logistyczna

Najpierw wytrenujemy model na zbiorze uczącym zawierającym wszystkie zmienne.

```

mlr.model.basic <- multinom(Type ~ ., data = train.data)

basic.ml.train.pred <- predict(mlr.model.basic, newdata = train.data, type = "class")
basic.ml.test.pred <- predict(mlr.model.basic, newdata = test.data, type = "class")

```

Wyznamy dla tego modelu macierze pomyłek i wartości błędów klasyfikacji.

| | 1 | 2 | 3 | | 1 | 2 | 3 |
|------------------|----|----|----|-------------------|----|----|----|
| 1 | 36 | 0 | 0 | 1 | 22 | 0 | 1 |
| 2 | 0 | 51 | 0 | 2 | 1 | 18 | 1 |
| 3 | 0 | 0 | 31 | 3 | 0 | 0 | 17 |
| (a) Zbiór uczący | | | | (b) Zbiór testowy | | | |

Tabela 15: Macierze pomyłek dla regresji wieloklasowej — wszystkie cechy.

Błędy klasyfikacji to kolejno 0 i 0.05.

Powtórzmy teraz powyższe dla wybranego podzbioru naszych danych.

```

mlr.model.subset <- multinom(Type ~ ., data = train.subset)

mlr.subset.train.pred <- predict(mlr.model.subset, train.subset, type = "class")
mlr.subset.test.pred <- predict(mlr.model.subset, test.subset, type = "class")

```

Błędy klasyfikacji to kolejno 0.0847458 i 0.1.

Ponownie skorzystamy z pakietu `caret`, by stworzyć model stunigowany. Wyznamy dla niego macierze pomyłek i błędy klasyfikacji.

| | 1 | 2 | 3 | | 1 | 2 | 3 |
|---|----|----|----|---|----|----|----|
| 1 | 34 | 2 | 0 | 1 | 19 | 4 | 0 |
| 2 | 3 | 45 | 3 | 2 | 1 | 19 | 0 |
| 3 | 0 | 2 | 29 | 3 | 0 | 1 | 16 |

(a) Zbior uczacy (b) Zbior testowy

Tabela 16: Macierze pomylek dla wieloklasowej regresji — wybrane cechy.

```
model <- train(Type ~ ., data = train.data, method = "multinom",
               trControl = cv)
```

| | 1 | 2 | 3 | | 1 | 2 | 3 |
|---|----|----|----|---|----|----|----|
| 1 | 35 | 1 | 0 | 1 | 22 | 1 | 0 |
| 2 | 0 | 51 | 0 | 2 | 1 | 18 | 1 |
| 3 | 0 | 0 | 31 | 3 | 0 | 0 | 17 |

(a) Zbior uczacy (b) Zbior testowy

Tabela 17: Macierze pomylek dla klasyfikatora regresji wieloklasowej — model stuningowany.

Błędy klasyfikacji w tym przypadku to kolejno 0.0084746 i 0.05.

Powtórzymy teraz podobną jak wcześniej ocenę klasyfikacji dla stuningowanego modelu regresji wieloklasowej.

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

mlr.predictor <- function(formula, data)
{ train(formula, data = data, method = "multinom", trControl = cv)}

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

mlr.error.boot <- errorest(Type~., wine[, c(1, 2, 7, 8)],
                          model=mlr.predictor,
                          predict=predictor, estimator="boot",
                          est.param=control.errorest(nboot = 25))

mlr.error.632 <- errorest(Type~., wine[, c(1, 2, 7, 8)],
                          model=mlr.predictor,
                          predict=predictor, estimator="632plus",
                          est.param=control.errorest(nboot = 25))
```

Błędy predykcji wyniosły 0.0898876, 0.1018866 oraz 0.0934978.

3.6 Podsumowanie

Porównaliśmy ze sobą 4 klasyfikatory. Dla każdego z nich badaliśmy jaki wpływ na dokładność ich predykcji ma zmiana charakterystycznych dla nich parametrów czy zmiany w zbiorze uczącym (klasyfikatory trenowaliśmy na zbiorach, które albo zawierały wszystkie zmienne, albo te wybrane przez nas, które wprowadziły najlepszy podział na zmienne).

- Dla metody k najbliższych sąsiadów najlepsze rezultaty otrzymaliśmy dla $k = 9$ i zbioru uczącego o zawężonej liczbie cech.
- Dla metody drzew decyzyjnych najlepszy okazała się wartość parametru $cp = 0$ i wyuczenie modelu na zbiorze z wybranymi przez nas cechami.
- Dla naiwnego klasyfikatora bayesowskiego najlepsze efekty otrzymaliśmy, gdy wyuczyliśmy model na zbiorze uczącym zawierającym wszystkie zmienne.
- Dla wieloklasowej regresji logistycznej najefektywniejsze okazało się wyuczenie modelu na zbiorze zawierającym wszystkie zmienne.

Błędy predykcji dla 5-krotnej walidacji krzyżowej, metody bootstrap oraz .632+ wyglądają następująco — patrz tabela.

| Metoda | KNN | Drzewa decyzyjne | Naiwny klasyfikator bayesowski | Wieloklasowa regresja logistyczna |
|-----------|-----------|------------------|--------------------------------|-----------------------------------|
| CV | 0.0842697 | 0.1067416 | 0.0337079 | 0.0898876 |
| Bootstrap | 0.1011415 | 0.1119916 | 0.0276237 | 0.1018866 |
| .632+ | 0.0822332 | 0.0940631 | 0.0195704 | 0.0934978 |

Tabela 18: Wartości błędów predykcji.

Możemy zauważyć, że najlepszą metodą okazał się naiwny klasyfikator bayesowski. Najgorzej natomiast poradziły sobie drzewa decyzyjne.