Eksploracja danych

true

2020-03-11

Spis treści

Wstęp

O książce

Niniejsza książka powstała na bazie doświadczeń autora, a głównym jej celem jest przybliżenie czytelnikowi podstaw z dziedziny *Data mining* studentom kierunku *Matematyka* Politechniki Lubelskiej. Będzie łączyć w sobie zarówno treści teoretyczne związane z przedstawianymi etapami eksploracji danych i budową modeli, jak i praktyczne wskazówki dotczące budowy modeli w środowisku **R** (R Core Team 2018). Podane zostaną również wskazówki, jak raportować wyniki analiz i jak dokonać właściwych ilustracji wyników. Bardzo użyteczny w napisaniu książki były pakiety programu R: **bookdown** (Xie 2018a), **knitr** (Xie 2018b) oraz pakiet **rmarkdown** (Allaire et al. 2018).

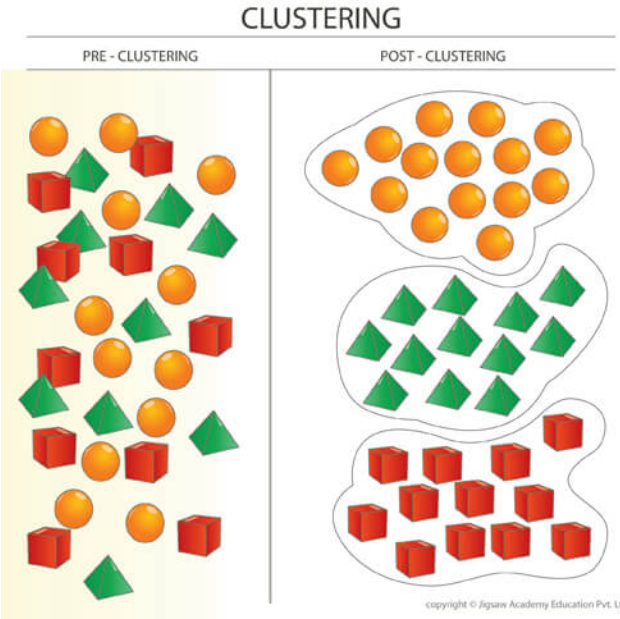
Zakres przedmiotu

Przedmiot *Eksploracja danych* będzie obejmował swoim zakresem eksplorację i wizualizację danych oraz uczenie maszynowe. Eksploracja danych ma na celu pozyskiwanie i systematyzację wiedzy pochodzącej z danych. Odbywa się ona głównie przy użyciu technik statystycznych, rachunku prawdopodobieństwa i metod z zakresu baz danych. Natomiast uczenie maszynowe, to gałąź nauki (obejmuje nie tylko statystykę, choć to na niej się głównie opiera) dotyczącej budowy modeli zdolnych do rozpoznawania wzorców, przewidywania wartości i klasyfikacji obiektów. Data mining to szybko rosnaca grupa metod analizy danych rozwijana nie tylko przez statystyków ale również przez biologów, genetyków, cybernetyków, informatyków, ekonomistów, osoby pracujace nad rozpoznawaniem obrazów i wiele innych grup zawodowych. W dzisiejszych czasch trudno sobie wyobrazić życie bez sztucznej inteligencji. Towarzyszy ona nam w codziennym, życiu kiedy korzystamy z telefonów komórkowych, wyszukiwarek internetowych, robotów sprzątających, automatycznych samochodów, nawigacji czy gier komputerowych. Lista ta jest niepełna i stale się wydłuża.

href="https://twitter.com/i/status/1091069356367200256">January 31, 2019

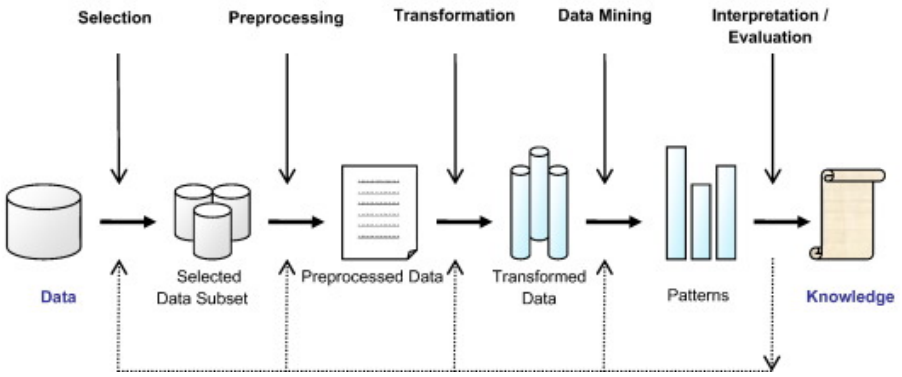
Zakres technik stosowanych w data mining

- statystyka opisowa
- wielowymiarowa analiza danych
- analiza szeregów czasowych
- analiza danych przestrzennych
- reguły asocjacji
- uczenie maszynowe[1], w tym:
  - klasyfikacja
  - predykcja
  - analiza skupień
  - *text mining*
- i wiele innych



Rysunek 1: Przykład nienadzorowanego uczenia maszynowego. *Źródło:*https://analyticstraining.com/cluster-analysis-for-business/

href="https://twitter.com/i/status/1097199751072690176">Ferbruary 17, 2019



Rysunek 2: Etapy eksploracji danych [@kavakiotis2017]

Etapy eksploracji danych

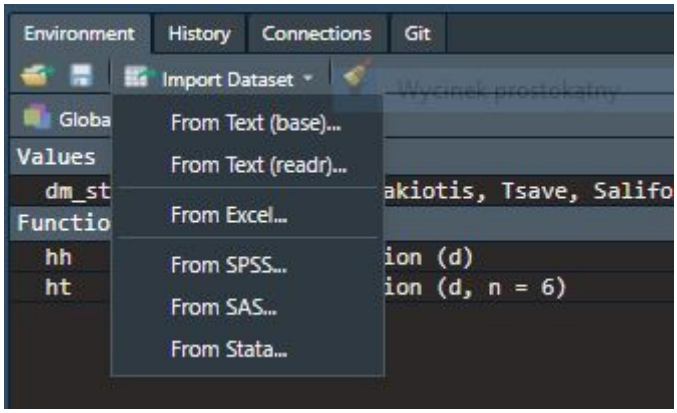
1. Czyszczenie danych - polega na usuwaniu braków danych, usuwaniu stałych zmiennych, imputacji braków danych oraz przygotowaniu danych do dalszych analiz.
2. Integracja danych - łączenie danych pochodzących z różnych źródeł.
3. Selekcja danych - wybór z bazy tych danych, które są potrzebne do dalszych analiz.
4. Transformacja danych - przekształcenie i konsolidacja danych do postaci przydatnej do eksploracji.
5. Eksploracja danych - zastosowanie technik wymienionych wcześniej w celu odnalezienia wzorców[2] i zależności.
6. Ewaluacja modeli - ocena poprawności modeli oraz wzorców z nich uzyskanych.
7. Wizualizacja wyników - graficzne przedstawienie odkrytych wzorców.
8. Wdrażanie modeli - zastosowanie wyznaczonych wzorców.

[1] ang. *machine learning*
[2] ang. *patterns*

Rozdział 1

Import danych

Środowisko **R** pozwala na import i export plików o różnych rozszerzeniach (`txt, csv, xls, xlsx, sav, xpt, dta`, itd.)[1]. W tym celu czasami trzeba zainstalować pakiety rozszerzające podstawowe możliwości R-a. Najnowsza[2] wersja programu RStudio (v. 1.1.463)[3] pozwala na wczytanie danych z popularnych źródeł za pomocą GUI.



Rysunek 1.1: Narzędzie do importu plików programu RStudio

Jeśli dane są zapisane w trybie tekstowym (np. `txt`, `csv`), to wczytujemy je w następujący sposób

```
dane1 <- read.table("data/dane1.txt", header = T)
head(dane1)
```

[1] póki co nie jest mi znana funkcja pozwalająca na import plików programu Statistica
[2] na dzień 19.02.2019
[3] istnieją rownież nowsze wersje deweloperskie

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
dane2 <- read.csv2("data/dane1.csv", header = T)
head(dane2)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
# funkcja pakietu readr wczytuje plik jako ramkę danych w formacie tibble
# pakiet readr jest częsią większego pakietu tidyverse,
# który został wczytany wczsniej
dane3 <- read_csv2("data/dane1.csv")
dane3
```

```
## # A tibble: 150 x 5
##    Sepal.Length Sepal.Width Petal.Length Petal.Width Species
##           <dbl>       <dbl>        <dbl>       <dbl> <chr>
##  1          5.1         3.5          1.4         0.2 setosa
##  2          4.9         3            1.4         0.2 setosa
##  3          4.7         3.2          1.3         0.2 setosa
##  4          4.6         3.1          1.5         0.2 setosa
##  5          5           3.6          1.4         0.2 setosa
##  6          5.4         3.9          1.7         0.4 setosa
##  7          4.6         3.4          1.4         0.3 setosa
##  8          5           3.4          1.5         0.2 setosa
##  9          4.4         2.9          1.4         0.2 setosa
## 10          4.9         3.1          1.5         0.1 setosa
## # ... with 140 more rows
```

Jeśli dane są przechowywane w pliku Excel (np. `xlsx`), to importujemy je za pomocą funkcji `read_excel` pakietu `readxl`. Domyślnie jest wczytywany arkusz pierwszy ale jeśli zachodzi taka potrzeba, to można ustalić, który arkusz pliku Excel ma być wczytany za pomocą paramteru `sheet`, np. `sheet=3`, co oznacza, że zostanie wczytany trzeci arkusz pliku.

Ponieważ w pliku `dane1.xlsx` braki danych zostały zakodowane znakami `BD` oraz `-`, to należy ten fakt przekazać funkcji, aby poprawnie wczytać braki da-

	A	B	C	D	E	F
1	Długość kielic	Szerokość	Długość pł	Szerokość	Gatunki	
2	5,1	3,5	1,4	0,2	setosa	
3	4,9	3	1,4	0,2	setosa	
4	4,7	3,2	1,3	0,2	setosa	
5	4,6	3,1	1,5	0,2	setosa	
6	5	3,6	1,4	0,2	setosa	
7	5,4	3,9	1,7	0,4	setosa	
8	-	-	1,4	0,3	setosa	
9	5	3,4	1,5	0,2	BD	
10	4,4	2,9	1,4	0,2	setosa	
11	4,9	3,1	1,5	0,1	setosa	
12	5,4	3,7	1,5	0,2	setosa	
13	4,8	3,4	1,6	0,2	setosa	

Rysunek 1.2: Fragment pliku Excel

nych. W przeciwnym przypadku zmienne zawierające braki tak kodowane, będą wczytane jako zmienne znakowe.

```
library(readxl)
dane4 <- read_excel("data/dane1.xlsx", na = c("BD", "-"))
dane4
```

```
## # A tibble: 150 x 5
##    `Długość kielich~ `Szerokość kieli~ `Długość płatka` `Szerokość płat~ Gatunki
##                <dbl>             <dbl>            <dbl>             <dbl> <chr>
##  1               5.1               3.5              1.4               0.2 setosa
##  2               4.9               3                1.4               0.2 setosa
##  3               4.7               3.2              1.3               0.2 setosa
##  4               4.6               3.1              1.5               0.2 setosa
##  5               5                 3.6              1.4               0.2 setosa
##  6               5.4               3.9              1.7               0.4 setosa
##  7              NA                NA                1.4               0.3 setosa
##  8               5                 3.4              1.5               0.2 <NA>
##  9               4.4               2.9              1.4               0.2 setosa
## 10               4.9               3.1              1.5               0.1 setosa
## # ... with 140 more rows
```

Istniej oczywiście jeszcze wiele innych fomatów danych, charakterystycznych dla programów, w których są traktowane jako domyślne.[4] W szczególny sposób

[4] do ich wczytywania stosujemy funkcje pakietu `foreign`

należy zwrócić uwagę na pliki o rozszerzeniu `RData` lub `rda`[5] oraz pliki `rds`. Pliki `rda` służą do przechowywania obiektów programu `R`. Mogą to być pliki danych ale również obiekty graficzne (typu wyniki funkcji `ggplot`), modele (np. wynik funkcji `lm()`), zdefiniowane funkcje i wszystkie inne obiekty, które da się zapisać w środowisku `R`. Ponadto pliki `rda` pozawalają na zapisanie wielu obiektów w jednym pliku. Pliki o rozszerzeniu `rds` mają podobną funkcję z tym, że pozwalają na przechowywanie tylko jednego obiektu.

```
# wszystkie wczytane wcześniej pliki zapisuje w jednym pliku
save(dane1, dane2, dane3, dane4, file = "data/dane.rda")
# plik rda został zapisany
list.files(path = "data/")
```

```
## [1] "algae.csv"    "Analysis.txt" "dane.rda"     "dane1.csv"    "dane1.txt"
## [6] "dane1.xlsx"   "dane4.rds"    "dane4.sav"
```

```
# usuwam dane ze środowiska R
rm(dane1, dane2, dane3, dane4)
# sprawdzam co jest wczytane do R
ls()
```

```
## character(0)
```

```
# wczytuję plik rda
load("data/dane.rda")
# jeszcze raz sprawdzam co jest wczytane do R
ls()
```

```
## [1] "dane1" "dane2" "dane3" "dane4"
```

Zapisując obiekty jako oddzielne pliki, można przy wczytywaniu nadawać im nazwy.

```
rm(dane1, dane2, dane3)
ls()
```

```
## [1] "dane4"
```

```
saveRDS(dane4, file = "data/dane4.rds")
nowe_dane <- readRDS("data/dane4.rds")
nowe_dane
```

```
## # A tibble: 150 x 5
##    `Długość kielich~ `Szerokość kieli~ `Długość płatka` `Szerokość płat~ Gatunki
##                <dbl>             <dbl>            <dbl>            <dbl> <chr>
##  1               5.1               3.5              1.4              0.2 setosa
##  2               4.9               3                1.4              0.2 setosa
##  3               4.7               3.2              1.3              0.2 setosa
##  4               4.6               3.1              1.5              0.2 setosa
```

[5] oznaczają to samo

```
##  5              5               3.6            1.4            0.2 setosa
##  6              5.4             3.9            1.7            0.4 setosa
##  7             NA              NA              1.4            0.3 setosa
##  8              5               3.4            1.5            0.2 <NA>
##  9              4.4             2.9            1.4            0.2 setosa
## 10              4.9             3.1            1.5            0.1 setosa
## # ... with 140 more rows
```

Oprócz wielu zalet takiego sposobu importu i eksportu danych jest jedna poważna wada, pliki te można odczytać jedynie za pomocą R. Osobiście polecam stosować do importu i eksportu danych plików w takich formatach, które mogą przeczytać wszyscy. Jak dotąd widać do importu różnych formatów danych potrzebujemy różnych funkcji, czasami nawet z różnych pakietów. Istnieje rozwiązanie tej niedogodności

```
library(rio)
dane1 <- import("data/dane1.txt")
head(dane1)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
dane2 <- import("data/dane1.csv", dec = ",")
# dane1.csv miały , jako znak rozdzielający cechę i mantysę liczb
# dlatego włączamy parametr dec
head(dane2)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
dane3 <- import("data/dane1.xlsx", na=c("BD","-"))
head(dane3)
```

```
##   Długość kielicha Szerokość kielicha Długość płatka Szerokość płatka Gatunki
## 1              5.1               3.5            1.4             0.2  setosa
## 2              4.9               3.0            1.4             0.2  setosa
## 3              4.7               3.2            1.3             0.2  setosa
## 4              4.6               3.1            1.5             0.2  setosa
## 5              5.0               3.6            1.4             0.2  setosa
```

```
## 6             5.4                3.9           1.7             0.4  setosa
```

```
dane4 <- import("data/dane4.rds")
dane4
```

```
## # A tibble: 150 x 5
##    `Długość kielich~ `Szerokość kieli~ `Długość płatka` `Szerokość płat~ Gatunki
##                <dbl>             <dbl>            <dbl>            <dbl> <chr>
##  1               5.1               3.5              1.4              0.2 setosa
##  2               4.9               3                1.4              0.2 setosa
##  3               4.7               3.2              1.3              0.2 setosa
##  4               4.6               3.1              1.5              0.2 setosa
##  5               5                 3.6              1.4              0.2 setosa
##  6               5.4               3.9              1.7              0.4 setosa
##  7              NA                NA                1.4              0.3 setosa
##  8               5                 3.4              1.5              0.2 <NA>
##  9               4.4               2.9              1.4              0.2 setosa
## 10               4.9               3.1              1.5              0.1 setosa
## # ... with 140 more rows
```

Lista możliwości jaką daje nam pakiet `rio` (Chan and Leeper 2018) jest niemal nieograniczona:[6]

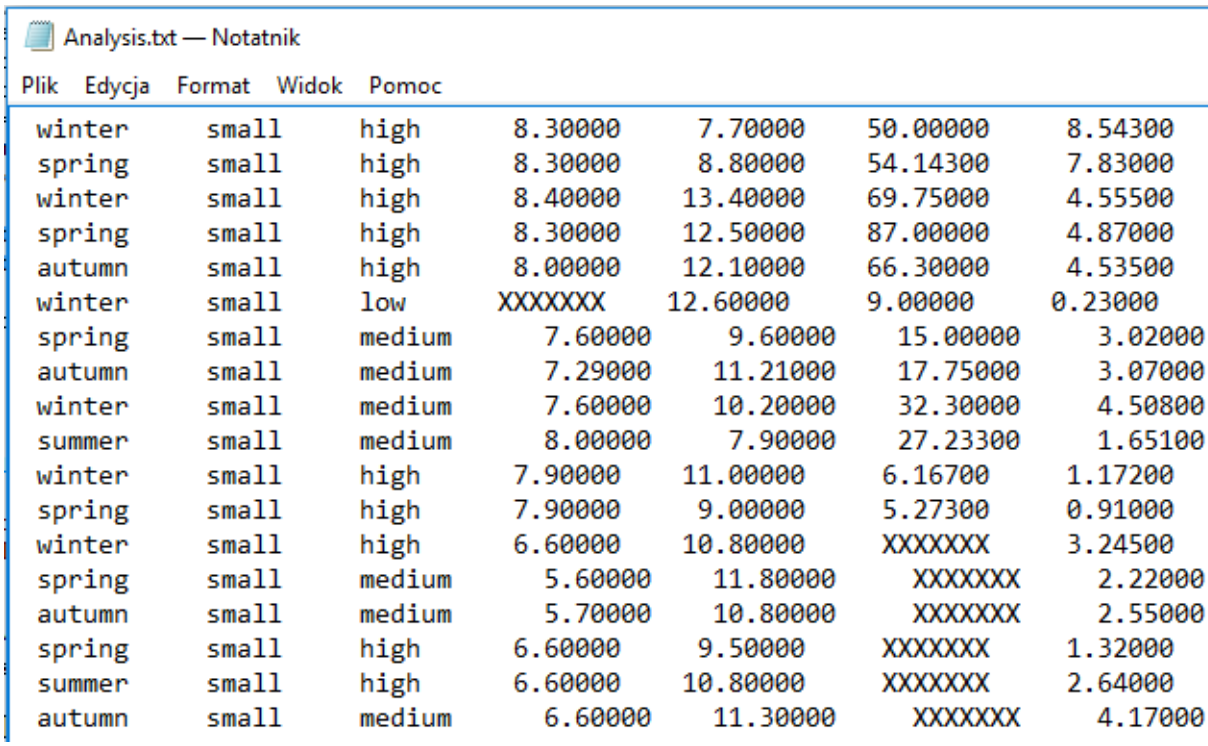
- Comma-separated data (.csv), using fread or, if fread = FALSE, read.table with row.names = FALSE and stringsAsFactors = FALSE
- Pipe-separated data (.psv), using fread or, if fread = FALSE, read.table with sep = '|', row.names = FALSE and stringsAsFactors = FALSE
- Tab-separated data (.tsv), using fread or, if fread = FALSE, read.table with row.names = FALSE and stringsAsFactors = FALSE
- SAS (.sas7bdat), using read_sas.
- SAS XPORT (.xpt), using read_xpt or, if haven = FALSE, read.xport.
- SPSS (.sav), using read_sav. If haven = FALSE, read.spss can be used.
- Stata (.dta), using read_dta. If haven = FALSE, read.dta can be used.
- SAS XPORT (.xpt), using read.xport.
- SPSS Portable Files (.por), using read_por.
- Excel (.xls and .xlsx), using read_excel. Use which to specify a sheet number. For .xlsx files, it is possible to set readxl = FALSE, so that read.xlsx can be used instead of readxl (the default).
- R syntax object (.R), using dget
- Saved R objects (.RData,.rda), using load for single-object .Rdata files. Use which to specify an object name for multi-object .Rdata files. This can be any R object (not just a data frame).
- Serialized R objects (.rds), using readRDS. This can be any R object (not just a data frame).
- Epiinfo (.rec), using read.epiinfo
- Minitab (.mtp), using read.mtp

[6] fragment pliku `help` funkcji `import`

- Systat (.syd), using read.systat
- "XBASE" database files (.dbf), using read.dbf
- Weka Attribute-Relation File Format (.arff), using read.arff
- Data Interchange Format (.dif), using read.DIF
- Fortran data (no recognized extension), using read.fortran
- Fixed-width format data (.fwf), using a faster version of read.fwf that requires a widths argument and by default in rio has stringsAsFactors = FALSE. If readr = TRUE, import will be performed using read_fwf, where widths should be: NULL, a vector of column widths, or the output of fwf_empty, fwf_widths, or fwf_positions.
- gzip comma-separated data (.csv.gz), using read.table with row.names = FALSE and stringsAsFactors = FALSE
- CSVY (CSV with a YAML metadata header) using read_csvy.
- Feather R/Python interchange format (.feather), using read_feather
- Fast storage (.fst), using read.fst
- JSON (.json), using fromJSON
- Matlab (.mat), using read.mat
- EViews (.wf1), using readEViews
- OpenDocument Spreadsheet (.ods), using read_ods. Use which to specify a sheet number.
- Single-table HTML documents (.html), using read_html. The data structure will only be read correctly if the HTML file can be converted to a list via as_list.
- Shallow XML documents (.xml), using read_xml. The data structure will only be read correctly if the XML file can be converted to a list via as_list.
- YAML (.yml), using yaml.load
- Clipboard import (on Windows and Mac OS), using read.table with row.names = FALSE
- Google Sheets, as Comma-separated data (.csv)

Przykład 1.1. Poniższa ilustracja przedstawia fragment pliku danych `Analysis.txt` zawierającego pewne błędy, które należy naprawić na etapie importu danych. Po pierwsze brakuje w nim nazw zmiennych (choć nie widać tego na rysunku). Poszczególne kolumny nazywają się następująco: `season`, `size`, `speed`, `mxPH`, `mnO2`, `Cl`, `NO3`, `NH4`, `oPO4`, `PO4`, `Chla`, `a1`, `a2`, `a3`, `a4`, `a5`, `a6`, `a7`. Naszym zadaniem jest import tego pliku z jednoczesną obsługą braków (braki danych są zakodowane przez `XXXXXXX`) oraz nadaniem nagłówków kolumn. Plik `Analisis.txt` jest umieszczony w kagalogu `data/`. Z racji, że plik dotyczy glonów, to dane zapiszemy pod nazwą `algae`.

```
algae <- import('data/Analysis.txt', header=F,
                dec='.',
                col.names=c('season','size','speed','mxPH','mnO2','Cl',
                            'NO3','NH4','oPO4','PO4','Chla','a1','a2',
                            'a3','a4','a5','a6','a7'),
                na.strings=c('XXXXXXX'))
```



Rysunek 1.3: Fragment pliku danych Analisis.txt

```
head(algae)
```

```
##   season  size  speed mxPH mnO2     Cl    NO3     NH4    oPO4     PO4 Chla   a1
## 1 winter small medium 8.00  9.8 60.800  6.238 578.000 105.000 170.000 50.0  0.0
## 2 spring small medium 8.35  8.0 57.750  1.288 370.000 428.750 558.750  1.3  1.4
## 3 autumn small medium 8.10 11.4 40.020  5.330 346.667 125.667 187.057 15.6  3.3
## 4 spring small medium 8.07  4.8 77.364  2.302  98.182  61.182 138.700  1.4  3.1
## 5 autumn small medium 8.06  9.0 55.350 10.416 233.700  58.222  97.580 10.5  9.2
## 6 winter small   high 8.25 13.1 65.750  9.248 430.000  18.250  56.667 28.4 15.1
##     a2   a3  a4   a5   a6  a7
## 1  0.0  0.0 0.0 34.2  8.3 0.0
## 2  7.6  4.8 1.9  6.7  0.0 2.1
## 3 53.6  1.9 0.0  0.0  0.0 9.7
## 4 41.0 18.9 0.0  1.4  0.0 1.4
## 5  2.9  7.5 0.0  7.5  4.1 1.0
## 6 14.6  1.4 0.0 22.5 12.6 2.9
```

```
summary(algae)
```

```
##     season              size              speed                mxPH
##  Length:200         Length:200         Length:200         Min.   :5.600
##  Class :character   Class :character   Class :character   1st Qu.:7.700
##  Mode  :character   Mode  :character   Mode  :character   Median :8.060
##                                                           Mean   :8.012
##                                                           3rd Qu.:8.400
##                                                           Max.   :9.700
##                                                           NA's   :1
##       mnO2              Cl               NO3              NH4
##  Min.   : 1.500   Min.   :  0.222   Min.   : 0.050   Min.   :    5.00
##  1st Qu.: 7.725   1st Qu.: 10.981   1st Qu.: 1.296   1st Qu.:   38.33
##  Median : 9.800   Median : 32.730   Median : 2.675   Median :  103.17
##  Mean   : 9.118   Mean   : 43.636   Mean   : 3.282   Mean   :  501.30
##  3rd Qu.:10.800   3rd Qu.: 57.824   3rd Qu.: 4.446   3rd Qu.:  226.95
##  Max.   :13.400   Max.   :391.500   Max.   :45.650   Max.   :24064.00
##  NA's   :2        NA's   :10        NA's   :2        NA's   :2
##       oPO4              PO4              Chla               a1
##  Min.   :  1.00   Min.   :  1.00   Min.   :  0.200   Min.   : 0.00
##  1st Qu.: 15.70   1st Qu.: 41.38   1st Qu.:  2.000   1st Qu.: 1.50
##  Median : 40.15   Median :103.29   Median :  5.475   Median : 6.95
##  Mean   : 73.59   Mean   :137.88   Mean   : 13.971   Mean   :16.92
##  3rd Qu.: 99.33   3rd Qu.:213.75   3rd Qu.: 18.308   3rd Qu.:24.80
##  Max.   :564.60   Max.   :771.60   Max.   :110.456   Max.   :89.80
##  NA's   :2        NA's   :2        NA's   :12
##        a2               a3               a4               a5
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.: 0.000
```

```
##  Median : 3.000   Median : 1.550   Median : 0.000   Median : 1.900
##  Mean   : 7.458   Mean   : 4.309   Mean   : 1.992   Mean   : 5.064
##  3rd Qu.:11.375   3rd Qu.: 4.925   3rd Qu.: 2.400   3rd Qu.: 7.500
##  Max.   :72.600   Max.   :42.800   Max.   :44.600   Max.   :44.400
##
##        a6               a7
##  Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 0.000   1st Qu.: 0.000
##  Median : 0.000   Median : 1.000
##  Mean   : 5.964   Mean   : 2.495
##  3rd Qu.: 6.925   3rd Qu.: 2.400
##  Max.   :77.600   Max.   :31.600
##
```

```
export(algae, file = "data/algae.csv")
```

Rozdział 2

Przygotowanie danych

Dane, które importujemy z zewnętrznego źródła najczęściej nie spełniają formatów obowiązujących w **R**. Często zmienne zawierają niedopuszczalne znaki szczególne, odstępy w nazwach, powtórzone nazwy kolumn, nazwy zmiennych zaczynające się od liczby, czy puste wiersze lub kolumny. Przed przystąpieniem do analizy zbioru należy rozważyć ewentualne poprawki nazw zmiennych, czy usunięcie pustych kolumn i wierszy. Niektórych czynności można dokonać już na etapie importu danych, stosując pewne pakiety oraz nowe funkcjonalności środowiska **RStudio**. W większości przypadków uchroni nas to od żmudnego przekształcania typów zmiennych. Oczywiście wszystkie te czynności czyszczenia danych można również dokonać już po imporcie danych, za pomocą odpowiednich komend **R**.

```
## przykładowe niepożądane nazwy zmiennych
test_df <- as.data.frame(matrix(rnorm(18),ncol = 6))
names(test_df) <- c("hIgHlo", "REPEAT VALUE", "REPEAT VALUE",
                    "% successful (2009)",  "abc@!*", "")
test_df
```

```
##        hIgHlo REPEAT VALUE REPEAT VALUE % successful (2009)     abc@!*
## 1 -0.08763204    0.9738085   -0.6274221           -1.283824  0.3841384
## 2  0.20395352   -0.7589554   -0.1586312            1.241683 -0.8977376
## 3 -0.65478115   -0.8860280    0.3505120           -1.528660  1.1383163
##
## 1 -0.1911037
## 2 -0.1132316
## 3 -0.4387924
```

```
## do poprawy nazw zmiennych użyjemy funkcji make.names
names(test_df) <- make.names(names(test_df))
test_df
```

```
##        hIgHlo REPEAT.VALUE REPEAT.VALUE X..successful..2009.     abc...
## 1 -0.08763204    0.9738085   -0.6274221            -1.283824  0.3841384
## 2  0.20395352   -0.7589554   -0.1586312             1.241683 -0.8977376
## 3 -0.65478115   -0.8860280    0.3505120            -1.528660  1.1383163
##            X
## 1 -0.1911037
## 2 -0.1132316
## 3 -0.4387924
```

Efekt końcowy choć skuteczny, to nie jest zadowalający. Czyszczenia nazw zmiennych można też dokonać stosując funkcję `clean_names` pakietu **janitor** (Firke 2018). Pozwala on również na usuwanie pustych wierszy i kolumn, znajdowanie zduplikowanych rekordów, itp.

```
library(janitor)
test_df %>% # aby na stałe zmienić nazwy zmiennych trzeba podstawienia
    clean_names()
```

```
##      h_ig_hlo repeat_value repeat_value_2 x_successful_2009        abc
## 1 -0.08763204    0.9738085     -0.6274221         -1.283824  0.3841384
## 2  0.20395352   -0.7589554     -0.1586312          1.241683 -0.8977376
## 3 -0.65478115   -0.8860280      0.3505120         -1.528660  1.1383163
##            x
## 1 -0.1911037
## 2 -0.1132316
## 3 -0.4387924
```

```
# przykładowe dane
x <- data.frame(w1=c(1,4,2,NA),w2=c(NA,2,3,NA), w3=c(1,NA,1,NA))
x
```

```
##   w1 w2 w3
## 1  1 NA  1
## 2  4  2 NA
## 3  2  3  1
## 4 NA NA NA
```

```
x %>% remove_empty("rows")
```

```
##   w1 w2 w3
## 1  1 NA  1
## 2  4  2 NA
## 3  2  3  1
```

2.1 Identyfikacja braków danych

Zanim usuniemy jakiekolwiek braki w zbiorze, powinniśmy je najpierw zidentyfikować, określić ich charakter, a dopiero potem ewentualnie podjąć decyzję o

uzupełnianiu braków.

```
algae <- rio::import("data/algae.csv")

# najprościej jest wywołać summary
summary(algae)
```

```
##     season              size              speed                mxPH
##  Length:200         Length:200         Length:200         Min.   :5.600
##  Class :character   Class :character   Class :character   1st Qu.:7.700
##  Mode  :character   Mode  :character   Mode  :character   Median :8.060
##                                                           Mean   :8.012
##                                                           3rd Qu.:8.400
##                                                           Max.   :9.700
##                                                           NA's   :1
##       mnO2              Cl               NO3              NH4
##  Min.   : 1.500   Min.   :  0.222   Min.   : 0.050   Min.   :    5.00
##  1st Qu.: 7.725   1st Qu.: 10.981   1st Qu.: 1.296   1st Qu.:   38.33
##  Median : 9.800   Median : 32.730   Median : 2.675   Median :  103.17
##  Mean   : 9.118   Mean   : 43.636   Mean   : 3.282   Mean   :  501.30
##  3rd Qu.:10.800   3rd Qu.: 57.824   3rd Qu.: 4.446   3rd Qu.:  226.95
##  Max.   :13.400   Max.   :391.500   Max.   :45.650   Max.   :24064.00
##  NA's   :2        NA's   :10        NA's   :2        NA's   :2
##       oPO4              PO4              Chla               a1
##  Min.   :  1.00   Min.   :  1.00   Min.   :  0.200   Min.   : 0.00
##  1st Qu.: 15.70   1st Qu.: 41.38   1st Qu.:  2.000   1st Qu.: 1.50
##  Median : 40.15   Median :103.29   Median :  5.475   Median : 6.95
##  Mean   : 73.59   Mean   :137.88   Mean   : 13.971   Mean   :16.92
##  3rd Qu.: 99.33   3rd Qu.:213.75   3rd Qu.: 18.308   3rd Qu.:24.80
##  Max.   :564.60   Max.   :771.60   Max.   :110.456   Max.   :89.80
##  NA's   :2        NA's   :2        NA's   :12
##        a2                a3               a4                a5
##  Min.   : 0.000   Min.   : 0.000   Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.: 0.000   1st Qu.: 0.000
##  Median : 3.000   Median : 1.550   Median : 0.000   Median : 1.900
##  Mean   : 7.458   Mean   : 4.309   Mean   : 1.992   Mean   : 5.064
##  3rd Qu.:11.375   3rd Qu.: 4.925   3rd Qu.: 2.400   3rd Qu.: 7.500
##  Max.   :72.600   Max.   :42.800   Max.   :44.600   Max.   :44.400
##
##        a6               a7
##  Min.   : 0.000   Min.   : 0.000
##  1st Qu.: 0.000   1st Qu.: 0.000
##  Median : 0.000   Median : 1.000
##  Mean   : 5.964   Mean   : 2.495
##  3rd Qu.: 6.925   3rd Qu.: 2.400
##  Max.   :77.600   Max.   :31.600
```

```
##
## wyświetl niekompletne wiersze
algae[!complete.cases(algae),] %>% head()
```

```
##    season  size  speed mxPH mnO2   Cl   NO3 NH4 oPO4 PO4 Chla   a1  a2  a3   a4
## 28 autumn small   high  6.8 11.1 9.00 0.630  20  4.0  NA  2.7 30.3 1.9 0.0  0.0
## 38 spring small   high  8.0   NA 1.45 0.810  10  2.5 3.0  0.3 75.8 0.0 0.0  0.0
## 48 winter small    low   NA 12.6 9.00 0.230  10  5.0 6.0  1.1 35.5 0.0 0.0  0.0
## 55 winter small   high  6.6 10.8   NA 3.245  10  1.0 6.5   NA 24.3 0.0 0.0  0.0
## 56 spring small medium  5.6 11.8   NA 2.220   5  1.0 1.0   NA 82.7 0.0 0.0  0.0
## 57 autumn small medium  5.7 10.8   NA 2.550  10  1.0 4.0   NA 16.8 4.6 3.9 11.5
##     a5  a6  a7
## 28 2.1 1.4 2.1
## 38 0.0 0.0 0.0
## 48 0.0 0.0 0.0
## 55 0.0 0.0 0.0
## 56 0.0 0.0 0.0
## 57 0.0 0.0 0.0
```

```
## policz niekompletne wiersze
nrow(algae[!complete.cases(algae),])
```

```
## [1] 16
```

```
## sprawdzenie liczby braków w wierszach
apply(algae, 1, function(x) sum(is.na(x)))
```

```
##   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
##   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
##   0   0   0   0   0   0   0   1   0   0   0   0   0   0   0   0   0   1   0   0
##  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
##   0   0   0   0   0   0   0   1   0   0   0   0   0   0   2   2   2   2   2   2
##  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
##   2   6   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
##  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
##   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
##   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   1   0   0   0   0
## 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
##   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160
##   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 161 162 163 164 165 166 167 168 169 170 171 172 173 174 175 176 177 178 179 180
##   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
## 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 199 200
##   0   0   0   1   0   0   0   0   0   0   0   0   0   0   0   0   0   0   6   0
```

Wiele ciekawych funkcji do eksploracji danych znajduje się w pakiecie **DMwR** (Torgo 2013), który został przygotowany przy okazji publikacji książki *Data Mining with R*.

```
## poszukiwanie wierszy zawierających wiele braków
## w tym przypadku próg wyświetlania ustawiony jest na 0.2
## czyli 20% wszystkich kolumn
library(DMwR)
manyNAs(algae)
```

```
##  62 199
##  62 199
```

```
## tworzenie zbioru pozbawionego wierszy zawierających wiele braków
algae2 <- algae[-manyNAs(algae), ]

## sprawdzamy liczbę wybrakowanych wierszy które pozostały
nrow(algae2[!complete.cases(algae2),])
```

```
## [1] 14
```

```
## usuwamy wszystkie wiersze z brakami
algae3 <- na.omit(algae)

## wyświetl wiersze z brakami
algae3[!complete.cases(algae3),] %>% head()
```

```
##  [1] season size   speed  mxPH   mnO2   Cl     NO3    NH4    oPO4   PO4
## [11] Chla   a1     a2     a3     a4     a5     a6     a7
## <0 rows> (or 0-length row.names)
```

```
## liczba pozostałych wybrakowanych wierszy
nrow(algae3[!complete.cases(algae3),])
```

```
## [1] 0
```

```
## można oczywiście też ręcznie usuwać wiersze (nie polecam)
algae4 <- algae[-c(62,199),]
```

Można też zbudować funkcję, która będzie usuwała braki danych wg naszego upodobania.

```
## najpierw budujemy funkcję i ją kompilujemy aby R mógł ja stosować
## parametr prog ustala próg odcięcia wierszy
czysc.dane <- function(dt, prog = 0){
    licz.braki <- apply(dt, 1, function(x) sum(is.na(x)))
    czyste.dt <- dt[!(licz.braki/ncol(dt)>prog), ]
    return(czyste.dt)
}
```

```
## potem ją możemy stosować
algae4 <- czysc.dane(algae)
nrow(algae4[!complete.cases(algae4),])
```

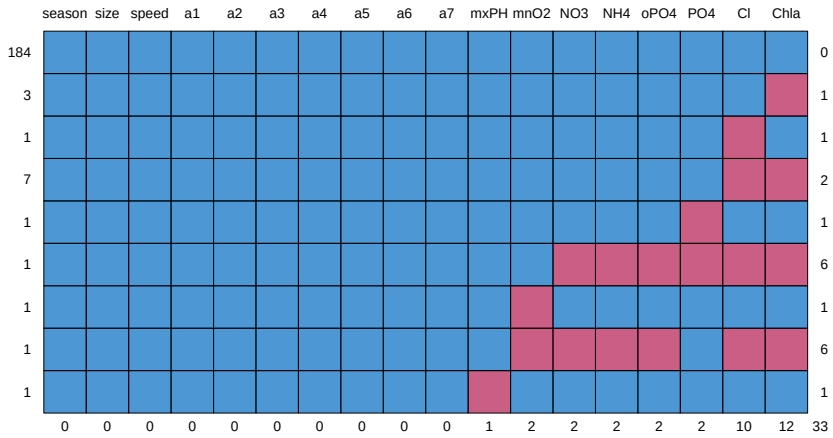
```
## [1] 0
```

```
## czyścimy wiersze, których liczba braków przekracza 20% wszystkich kolumn
algae5 <- czysc.dane(algae, prog = 0.2)
nrow(algae5[!complete.cases(algae5),])
```

```
## [1] 14
```

Bardzo ciekawym narzędziem do znajdowania braków danych jest funkcja `md.pattern` pakietu **mice** (van Buuren and Groothuis-Oudshoorn 2018). Wskazuje on ile braków występuje w ramach każdej zmiennej.

```
library(mice)
md.pattern(algae)
```



Rysunek 2.1: Na czerwono zaznaczone są zmienne, które zwierają braki danych. Liczba w wierszu po lewej stronie wykresu wskazuje ile wierszy w bazie ma daną charakterystykę, a liczba po prawej oznacza ile zmiennych było *wybrakowanych*

```
##     season size speed a1 a2 a3 a4 a5 a6 a7 mxPH mnO2 NO3 NH4 oPO4 PO4 Cl Chla
## 184      1    1     1  1  1  1  1  1  1  1    1    1   1   1    1   1  1    1
```

```
## 3        1    1     1  1  1  1  1  1  1  1    1    1  1  1    1  1  1    0
## 1        1    1     1  1  1  1  1  1  1  1    1    1  1  1    1  1  0    1
## 7        1    1     1  1  1  1  1  1  1  1    1    1  1  1    1  1  0    0
## 1        1    1     1  1  1  1  1  1  1  1    1    1  1  1    1  0  1    1
## 1        1    1     1  1  1  1  1  1  1  1    1    1  0  0    0  0  0    0
## 1        1    1     1  1  1  1  1  1  1  1    1    0  1  1    1  1  1    1
## 1        1    1     1  1  1  1  1  1  1  1    1    0  0  0    0  1  0    0
## 1        1    1     1  1  1  1  1  1  1  1    0    1  1  1    1  1  1    1
##          0    0     0  0  0  0  0  0  0  0    1    2  2  2    2  2 10   12
##
## 184  0
## 3    1
## 1    1
## 7    2
## 1    1
## 1    6
## 1    1
## 1    6
## 1    1
##     33
```

2.2 Zastępowanie braków danych

Zastępowanie braków danych (zwane także *imputacją danych*) jest kolejnym etapem procesu przygotowania danych do analiz. Nie można jednak wyróżnić uniwersalnego sposobu zastępowania braków dla wszystkich możliwych sytuacji. Wśród statystyków panuje przekonanie, że w przypadku wystąpienia braków danych można zastosować trzy strategie:

- nic nie robić z brakami - co wydaje się niedorzeczne ale wcale takie nie jest, ponieważ istnieje wiele modeli statystycznych (np. drzewa decyzyjne), które świetnie radzą sobie w sytuacji braków danych. Niestety nie jest to sposób, który można stosować zawsze, ponieważ są również modele wymagające kompletności danych jak na przykład sieci neuronowe.
- usuwać braki wierszami[1] - to metoda, która jest stosowana domyślnie w przypadku kiedy twórca modelu nie zadecyduje o innym sposobie obsługi luk. Metoda ta ma swoją niewątpliwą zaletę w postaci jasnej i prostej procedury, ale szczególnie w przypadku niewielkich zbiorów może skutkować obciążeniem estymatorów. Nie wiemy bowiem jaka wartość faktycznie jest przypisana danej cesze. Jeśli jest to wartość bliska np. średniej, to nie wpłynie znacząco na obciążenie estymatora wartości oczekiwanej. W przypadku, gdy różni się ona znacznie od średniej tej cechy, to estymator może już wykazywać obciążenie. Jego wielkość zależy również od liczby usuniętych elementów. Nie jest zalecane usuwanie wielu wierszy ze zbioru

[1] polega na usuwaniu wierszy zawierających braki

danych i na podstawie okrojonego zbioru wyciąganie wniosków o populacji, ponieważ próba jest wówczas znacząco inna niż populacja. Dodatkowo jeśli estymatory są wyznaczane na podstawie zbioru wyraźnie mniej licznego, to precyzja estymatorów wyrażona wariancją spada. Reasumując, jeśli liczba wierszy z brakującymi danymi jest niewielka w stosunku do całego zbioru, to usuwanie wierszy jest sensownym rozwiązaniem.

- uzupełnianie braków - to procedura polegająca na zastępowaniu braków różnymi technikami. Jej niewątpliwą zaletą jest fakt posiadania kompletnych danych bez konieczności usuwania wierszy. Niestety wiąże się to również z pewnymi wadami. Zbiór posiadający wiele braków uzupełnianych nawet bardzo wyrafinowanymi metodami może cechować się zaniżoną wariancją poszczególnych cech oraz tzw. przeuczeniem[2].

1. Uzupełnianie średnią - braki w zakresie danej zmiennej uzupełniamy średnią tej zmiennej przypadków uzupełnionych.

```
algae[is.na(algae$mxPH), ]
```

```
##    season  size speed mxPH mnO2 Cl  NO3 NH4 oPO4 PO4 Chla   a1 a2 a3 a4 a5 a6
## 48 winter small   low   NA 12.6  9 0.23  10    5   6  1.1 35.5  0  0  0  0  0
##    a7
## 48  0
```

```
m <- mean(algae$mxPH, na.rm = T)
algae[is.na(algae$mxPH), "mxPH"] <- m
algae[is.na(algae$mxPH), ]
```

```
##  [1] season size   speed  mxPH   mnO2   Cl     NO3    NH4    oPO4   PO4
## [11] Chla   a1     a2     a3     a4     a5     a6     a7
## <0 rows> (or 0-length row.names)
```

2. Uzupełnianie medianą - braki w zakresie danej zmiennej uzupełniamy medianą tej zmiennej przypadków uzupełnionych.

```
algae %>% filter(is.na(Chla)) %>% head
```

```
##   season  size  speed mxPH mnO2 Cl   NO3 NH4 oPO4  PO4 Chla   a1  a2  a3   a4
## 1 winter small   high  6.6 10.8 NA 3.245  10    1  6.5   NA 24.3 0.0 0.0  0.0
## 2 spring small medium  5.6 11.8 NA 2.220   5    1  1.0   NA 82.7 0.0 0.0  0.0
## 3 autumn small medium  5.7 10.8 NA 2.550  10    1  4.0   NA 16.8 4.6 3.9 11.5
## 4 spring small   high  6.6  9.5 NA 1.320  20    1  6.0   NA 46.8 0.0 0.0 28.8
## 5 summer small   high  6.6 10.8 NA 2.640  10    2 11.0   NA 46.9 0.0 0.0 13.4
## 6 autumn small medium  6.6 11.3 NA 4.170  10    1  6.0   NA 47.1 0.0 0.0  0.0
##   a5  a6 a7
## 1  0 0.0  0
## 2  0 0.0  0
## 3  0 0.0  0
```

[2] więcej o zjawisku przeuczenia w dalszej części książki

```
## 4  0 0.0  0
## 5  0 0.0  0
## 6  0 1.2  0
```

```
algae[is.na(algae$Chla), "Chla"] <- median(algae$Chla, na.rm = T)
```

3. Wypełnianie zmiennych typu wyliczeniowego, logicznego lub znakowego odbywa się najczęściej przez dobranie w miejsce brakującej wartości, elementu powtarzającego się najczęściej wśród obiektów obserwowanych. W pakiecie **DMwR** istnieje funkcja `centralImputation`, która wypełnia braki wartością centralną (w przypadku zmiennych typu liczbowego - medianą, a dla wartości logicznych, wyliczeniowych lub tekstowych - modą).

```
algae[48, "season"]
```

```
## [1] "winter"
```

```
algae[48, "season"] <- NA
algae.uzup <- centralImputation(algae)
algae.uzup[48,]
```

```
##    season  size speed     mxPH mnO2 Cl  NO3 NH4 oPO4 PO4 Chla   a1 a2 a3 a4 a5
## 48 winter small   low 8.011734 12.6  9 0.23  10    5   6  1.1 35.5  0  0  0  0
##    a6 a7
## 48  0  0
```

4. Jeszcze innym sposobem imputacji danych są algorytmy oparte o metodę k-najbliższych sąsiadów. Algorytm opiera się na prostej zasadzie, uzupełniania brakujących wartości medianą (w przypadku zmiennych ilościowych) lub modą (w przypadku zmiennych jakościowych) elementów, które są k-tymi najbliższymi sąsiadami w metryce

$$d(x, y) = \sqrt{\sum_{i=1}^{p} \delta_i(x_i, y_i)}, \tag{2.1}$$

gdzie δ_i jest odległością pomiędzy dwoma elementami ze względu na i-tą cech, określoną następująco

$$\delta_i(v_1, v_2) = \begin{cases} 1, & \text{jeśli zmienna jest jakościowa i } v_1 \neq v_2 \\ 0, & \text{jeśli zmienna jest jakościowa i } v_1 = v_2 \\ (v_1 - v_2)^2, & \text{jeśli zmienna jest ilościowa.} \end{cases} \tag{2.2}$$

Odległości są mierzone dla zmiennych standaryzowanych. Istnieje też odmiana z wagami, które maleją wraz ze wzrostem odległości pomiędzy sąsiadem a uzupełnianym elementem (np. $w(d) = \exp(d)$).

```
algae[48, ]
```

```
##    season  size speed     mxPH mnO2 Cl  NO3 NH4 oPO4 PO4 Chla   a1 a2 a3 a4 a5
```

```
## 48   <NA> small   low 8.011734 12.6  9 0.23  10    5   6  1.1 35.5  0  0  0  0
##    a6 a7
## 48  0  0
```

```
algae <- algae %>%
    mutate_if(is.character, as.factor)
algae.uzup <- knnImputation(algae, k = 5, scale = F, meth = "median")
algae.uzup[48,]
```

```
##    season  size speed     mxPH mnO2 Cl  NO3 NH4 oPO4 PO4 Chla   a1 a2 a3 a4 a5
## 48 summer small   low 8.011734 12.6  9 0.23  10    5   6  1.1 35.5  0  0  0  0
##    a6 a7
## 48  0  0
```

Istnieją również dużo bardziej złożone algorytmy imputacji danych oparte na bardziej wyrafinowanych technikach, takich jak: predykcja modelami liniowymi, nieliniowymi, analiza dyskryminacyjna, drzewa klasyfikacyjne. Dwa najbardziej znane pakiety zawierające funkcje do imputacji w sposób złożony, to **Amelia** i **mice**.

Imputacja danych z zastosowaniem pakietu **mice** wymaga podjęcia kilku decyzji przed przystąpieniem do uzupełniania danych:

1. Czy dane są MAR (ang. *Missing At Random*) czy MNAR (ang. *Missing Not At Random*), co oznacza, że musimy się zastanowić jakie mogły być źródła braków danych, przypadkowe czy systematyczne?
2. Należy się zdecydować na formę imputacji, określając strukturę zależności pomiędzy cechami oraz rozkład błędu danej cechy?
3. Wybrać zbiór danych, który posłuży nam za predyktory w imputacji (nie mogą zawierać braków).
4. Określenie, które niepełne zmienne są funkcjami innych wybrakowanych zmiennych.
5. Określić w jakiej kolejności dane będą imputowane.
6. Określić parametry startowe imputacji (liczbę iteracji, warunek zbieżności).
7. Określić liczę imputowanych zbiorów.

Ad 1. Wyróżniamy następujące rodzaje braków danych:

- MCAR (ang. *Missing Completely At Random*) - z definicji to braki, których pojawienie się jest kompletnie losowe. Przykładowo gdy osoba poproszona o wypełnienie wieku w ankiecie będzie rzucać monetą czy wypełnić tą zmienną.
- MAR - oznacza, że obserwowane wartości i wybrakowane mają inne rozkłady ale da się je oszacować na podstawie danych obserwowanych. Przykładowo ciśnienie tętnicze u osób, które nie wypełniły tej wartości jest wyższe niż u osób, które wpisały swoje ciśnienie. Okazuje się, że osoby starsze z nadciśnieniem nie wypełniały ankiety w tym punkcie.

- MNAR - jeśli nie jest spełniony warunek MCAR i MAR, wówczas brak ma charakter nielosowy. Przykładowo respondenci osiągający wyższe zarobki sukcesywnie nie wypełniają pola "zarobki" i dodatkowo nie ma w ankiecie zmiennych, które pozwoliłyby nam ustalić, jakie to osoby.

Ad 2. Decyzja o algorytmie imputacji wynika bezpośrednio ze skali w jakiej jest mierzona dana zmienna. Ze względu na rodzaj cechy używać będziemy następujących metod:

Każdy z czterech typów danych ma swój domyślny algorytm przeznaczony do imputacji:

- zmienna ilościowa - `pmm`
- zmienna dychotomiczna (stany 0 lub 1) - `logreg`
- zmienna typu wyliczeniowego (nieuporządkowana) - `polyreg`
- zmienna typu wyliczeniowego (uporządkowana) - `polr`

Niewątpliwą zaletą metody `pmm` jest to, że wartości imputowane są ograniczone jedynie do obserwowanych wartości. Metody `norm` i `norm.nob` uzupełniają brakujące wartości w oparciu o model liniowy. Są one szybkie i efektywne w przypadku gdy reszty modelu są zbliżone rozkładem do normalności. Druga z tych technik nie bierze pod uwagę niepewności związanej z modelem imputującym. Metoda `2L.norm` opiera się na dwupoziomowym heterogenicznym modelu liniowym (skupienia są włączone jako efekt do modelu). Technika `polyreg` korzysta z funkcji `multinom` pakietu **nnet** tworzącej model wielomianowy. `polr` opiera się o proporcjonalny model logitowy z pakietu **MASS**. `lda` to model dyskryminacyjny klasyfikujący obiekty na podstawie prawdopodobieństw *a posteriori*. Metoda `sample` zastępuje braki losowa wybranymi wartościami spośród wartości obserwowanych.

Ad 3. Do ustalenia predyktorów w modelu `mice` służy funkcja `predictorMatrix`. Po pierwsze wyświetla ona domyślny układ predyktorów włączanych do modelu. Można go dowolnie zmienić i podstawić do modelu imputującego dane parametrem `predictorMatrix`. Zera występujące w kolejnych wierszach macierzy predyktorów oznaczają pominięcie tej zmiennej przy imputacji innej zmiennej. Jeśli dodatkowo chcemy by jakaś zmienna nie była imputowana, to oprócz usunięcia jej z listy predyktorów, należy wymazać ją z listy metod predykcji (`method`).

Ogólne zalecenia co do tego jakie zmienne stosować jako predyktory jest takie, żeby brać ich jak najwięcej. Spowoduje to, że bardziej prawdopodobny staje się brak typu MAR a nie MNAR. Z drugiej jednak strony, nierzadko zbiory zawierają olbrzymią liczbę zmiennych i włączanie ich wszystkich do modelu imputującego nie będzie miało sensu.

Zalecenia doboru zmiennych są następujące:

- weź wszystkie te zmienne, które są włączane do modelu właściwego, czyli tego za pomocą którego chcesz poznać strukturę zależności;
- czasem do modelu imputującego należy też włączyć interakcje zmiennych z modelu właściwego;

Tabela 2.1: Zestaw metod imputacji danych stosowanych w pakiecie **mice**

method	type	description
pmm	any	Predictive.mean.matching
midastouch	any	Weighted predictive mean matching
sample	any	Random sample from observed values
cart	any	Classification and regression trees
rf	any	Random forest imputations
mean	numeric	Unconditional mean imputation
norm	numeric	Bayesian linear regression
norm.nob	numeric	Linear regression ignoring model error
norm.boot	numeric	Linear regression using bootstrap
norm.predict	numeric	Linear regression, predicted values
quadratic	numeric	Imputation of quadratic terms
ri	numeric	Random indicator for nonignorable data
logreg	binary	Logistic regression
logreg.boot	binary	Logistic regression with bootstrap
polr	ordered	Proportional odds model
polyreg	unordered	Polytomous logistic regression
lda	unordered	Linear discriminant analysis
2l.norm	numeric	Level-1 normal heteroscedastic
2l.lmer	numeric	Level-1 normal homoscedastic, lmer
2l.pan	numeric	Level-1 normal homoscedastic, pan
2l.bin	binary	Level-1 logistic, glmer
2lonly.mean	numeric	Level-2 class mean
2lonly.norm	numeric	Level-2 class normal
2lonly.pmm	any	Level-2 class predictive mean matching

- dodaj zmienne, które mogą mieć wpływ na wybrakowane cechy;
- włącz zmienne istotnie podnoszące poziom wyjaśnionej wariancji modelu;
- na koniec usuń te zmienne spośród predyktorów, które same zawierają zbyt wiele braków.

Ad 4-7. Decyzje podejmowane w tych punktach zależą istotnie od analizowanego zbioru i będą przedmiotem oddzielnych analiz w kontekście rozważanych zbiorów i zadań.

Przykład 2.1. Dokonamy imputacji zbioru `airquality` z wykorzystaniem pakietów **mice** i **VIM** (Templ et al. 2019)

```
data <- airquality
summary(data)
```

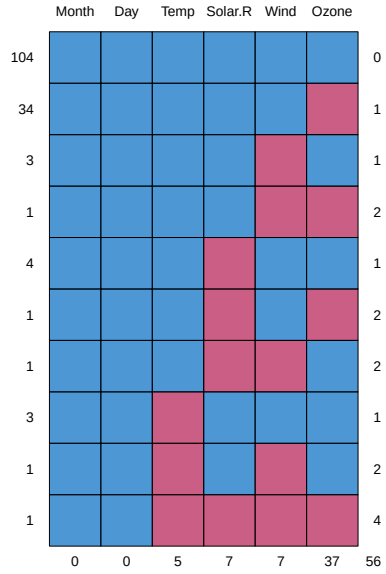
```
##      Ozone           Solar.R          Wind             Temp      
##  Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :56.00  
##  1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:72.00  
##  Median : 31.50   Median :205.0   Median : 9.700   Median :79.00  
##  Mean   : 42.13   Mean   :185.9   Mean   : 9.958   Mean   :77.88  
##  3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00  
##  Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00  
##  NA's   :37       NA's   :7                                       
##      Month            Day       
##  Min.   :5.000   Min.   : 1.0  
##  1st Qu.:6.000   1st Qu.: 8.0  
##  Median :7.000   Median :16.0  
##  Mean   :6.993   Mean   :15.8  
##  3rd Qu.:8.000   3rd Qu.:23.0  
##  Max.   :9.000   Max.   :31.0  
##
```

```
# tworzymy dodatkowe braki danych
data[4:10,3] <- rep(NA,7)
data[1:5,4] <- NA
summary(data)
```

```
##      Ozone           Solar.R          Wind             Temp      
##  Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :57.00  
##  1st Qu.: 18.00   1st Qu.:115.8   1st Qu.: 7.400   1st Qu.:73.00  
##  Median : 31.50   Median :205.0   Median : 9.700   Median :79.00  
##  Mean   : 42.13   Mean   :185.9   Mean   : 9.806   Mean   :78.28  
##  3rd Qu.: 63.25   3rd Qu.:258.8   3rd Qu.:11.500   3rd Qu.:85.00  
##  Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00  
##  NA's   :37       NA's   :7       NA's   :7        NA's   :5      
##      Month            Day       
##  Min.   :5.000   Min.   : 1.0  
##  1st Qu.:6.000   1st Qu.: 8.0
```

```
##  Median :7.000   Median :16.0
##  Mean   :6.993   Mean   :15.8
##  3rd Qu.:8.000   3rd Qu.:23.0
##  Max.   :9.000   Max.   :31.0
##
```

```
md.pattern(data)
```

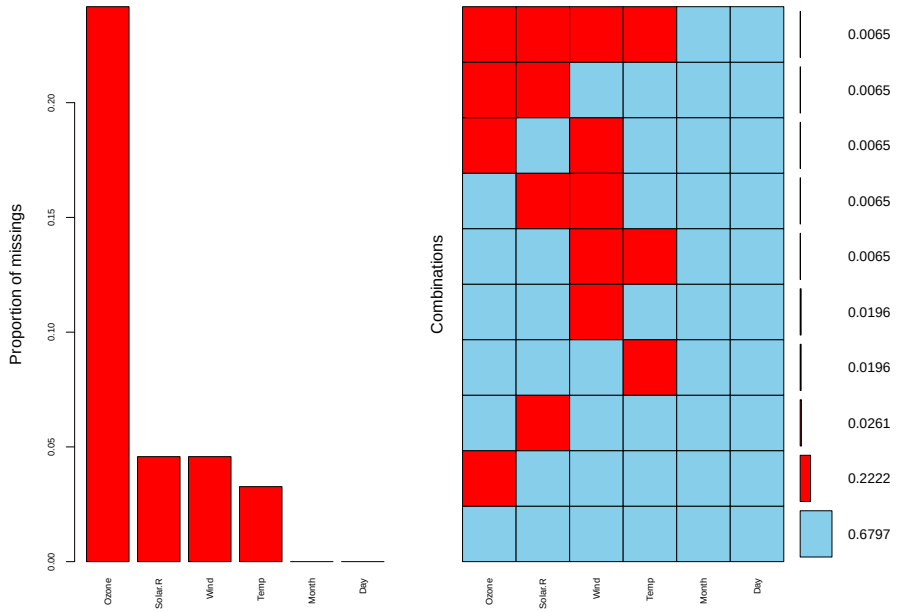



```
##     Month Day Temp Solar.R Wind Ozone
## 104     1   1    1       1    1     1  0
## 34      1   1    1       1    1     0  1
## 3       1   1    1       1    0     1  1
## 1       1   1    1       1    0     0  2
## 4       1   1    1       0    1     1  1
## 1       1   1    1       0    1     0  2
## 1       1   1    1       0    0     1  2
## 3       1   1    0       1    1     1  1
## 1       1   1    0       1    0     1  2
## 1       1   1    0       0    0     0  4
##         0   0    5       7    7    37 56
```

Do ilustracji braków danych można zastosować funkcje pakietu **VIM**.

```
library(VIM)
aggr(data, numbers=TRUE,
     sortVars=TRUE,
```

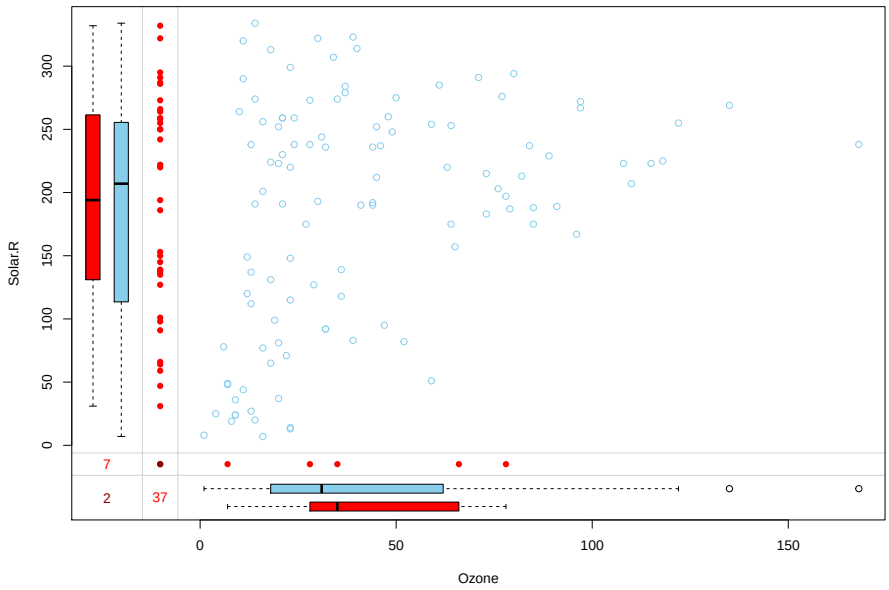
```
     labels=names(data),
     cex.axis=.7)
```




```
##
##  Variables sorted by number of missings:
##  Variable      Count
##     Ozone 0.24183007
##   Solar.R 0.04575163
##      Wind 0.04575163
##      Temp 0.03267974
##     Month 0.00000000
##       Day 0.00000000
```

Tak przedstawia się wykres rozrzutu zmiennych `Ozone` i `Solar.R` z uwzględnieniem położenia braków danych.

```
marginplot(data[c(1,2)])
```



Dokonamy imputacji metodą `pmm`.

```
tempData <- mice(data,
                 maxit=50,
                 meth='pmm',
                 seed=44,
                 printFlag = F)
summary(tempData)
```

```
## Class: mids
## Number of multiple imputations:  5
## Imputation methods:
##   Ozone Solar.R    Wind    Temp   Month     Day
##   "pmm"   "pmm"   "pmm"   "pmm"      ""      ""
## PredictorMatrix:
##         Ozone Solar.R Wind Temp Month Day
## Ozone       0       1    1    1     1   1
## Solar.R     1       0    1    1     1   1
## Wind        1       1    0    1     1   1
## Temp        1       1    1    0     1   1
## Month       1       1    1    1     0   1
## Day         1       1    1    1     1   0
```

Ponieważ, funkcja `mice` domyślnie dokonuje 5 kompletnych imputacji, możemy się przekonać jak bardzo różnią się poszczególne imputacje i zdecydować się na

jedną z nich.

```
head(tempData$imp$Ozone)
```

```
##     1  2  3  4  5
## 5  28 29 20 18 45
## 10 23 13  9 13 12
## 25 18 14 18 14  6
## 26 12 37  1 20 28
## 27 23  9 13 12 13
## 32 45 41 20 23 46
```

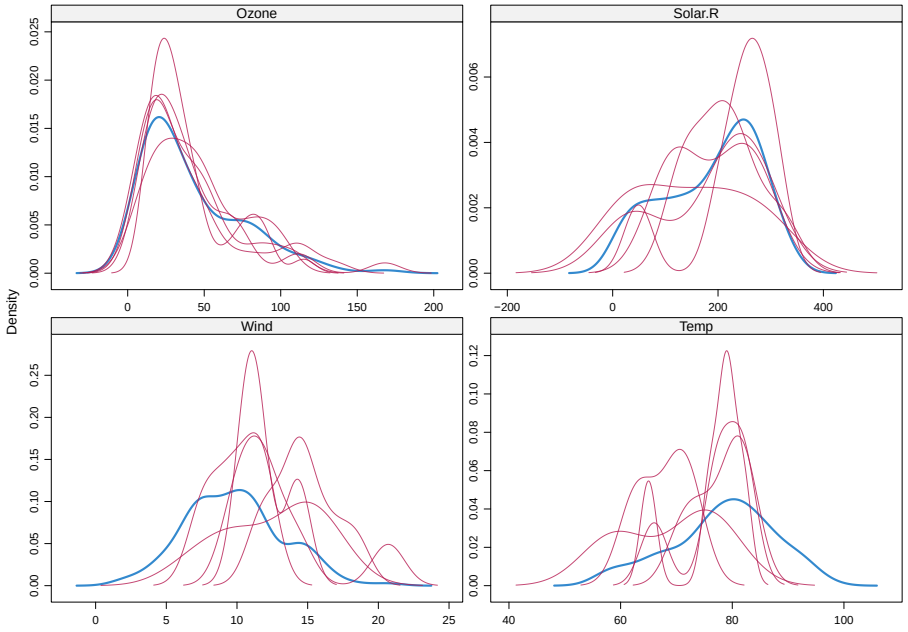
Ostatecznie imputacji dokonujemy wybierając jeden z zestawów danych uzupełniających (np. pierwszy).

```
completedData <- mice::complete(tempData, 1)
summary(completedData)
```

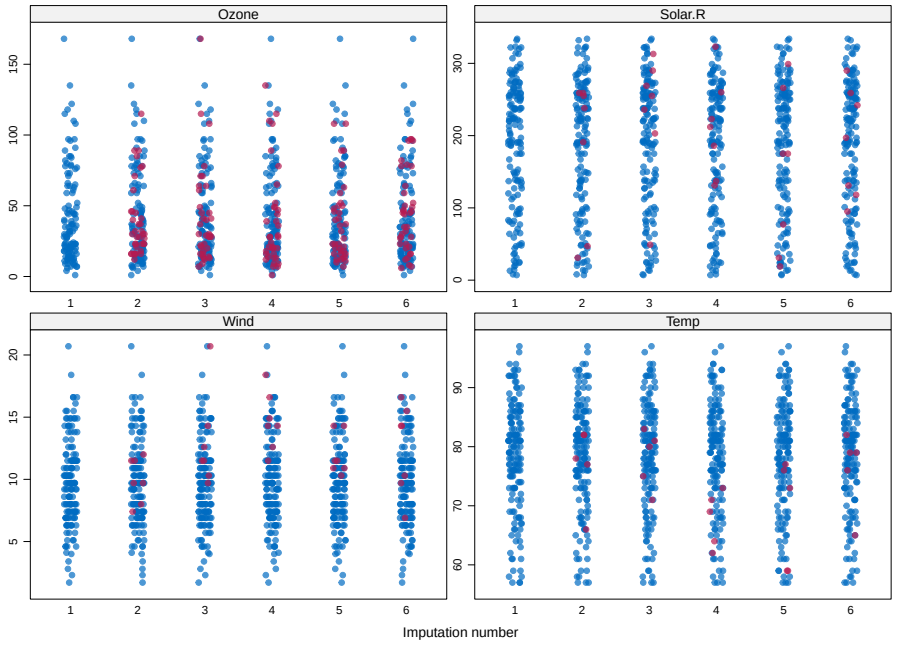
```
##      Ozone           Solar.R           Wind             Temp
##  Min.   :  1.00   Min.   :  7.0   Min.   : 1.700   Min.   :57.00
##  1st Qu.: 18.00   1st Qu.:115.0   1st Qu.: 7.400   1st Qu.:73.00
##  Median : 30.00   Median :207.0   Median : 9.700   Median :79.00
##  Mean   : 41.46   Mean   :185.8   Mean   : 9.814   Mean   :78.24
##  3rd Qu.: 61.00   3rd Qu.:259.0   3rd Qu.:11.500   3rd Qu.:85.00
##  Max.   :168.00   Max.   :334.0   Max.   :20.700   Max.   :97.00
##      Month            Day
##  Min.   :5.000   Min.   : 1.0
##  1st Qu.:6.000   1st Qu.: 8.0
##  Median :7.000   Median :16.0
##  Mean   :6.993   Mean   :15.8
##  3rd Qu.:8.000   3rd Qu.:23.0
##  Max.   :9.000   Max.   :31.0
```

Za pomocą funkcji pakietu `mice` możemy również przedstawić graficznie gdzie i jak zostały uzupełnione dane.

```
densityplot(tempData, ~Ozone+Solar.R+Wind+Temp)
```




```
stripplot(tempData, Ozone+Solar.R+Wind+Temp~.imp, pch = 20, cex = 1.2)
```



Rozdział 3

Podział metod data mining

3.1 Rodzaje wnioskowania

Data mining to zestaw metod pozyskiwania wiedzy na podstawie danych. Ową wiedzę zdobywamy w procesie wnioskowania na podstawie modeli. Wnioskowanie możemy podzielić na dedukcyjne i indukcyjne. I tak z wnioskowaniem dedukcyjnym mamy do czynienia wówczas, gdy na podstawie obecnego stanu wiedzy potrafimy odpowiedzieć na postawione pytanie dotyczące nowej wiedzy, stosując reguły wnioskowania. O wnioskowaniem indukcyjnym powiemy, że jest to metoda pozyskiwania wiedzy na podstawie informacji ze zbioru uczącego. Znajduje ono szerokie zastosowanie w data mining i charakteryzuje się omylnością, ponieważ nawet najlepiej nauczony model na zbiorze uczącym nie zapewnia nam prawdziwości odpowiedzi w przypadku nowych danych, a jedynie je uprawdopodabnia. Esencją wnioskowania indukcyjnego w zakresie data mining, jest poszukiwanie na podstawie danych uczących modelu charakteryzującego się najlepszymi właściwościami predykcyjnymi i dającego się zastosować do zupełnie nowego zbioru danych.

Każdy proces uczenia z wykorzystaniem wnioskowania indukcyjnego składa się z następujących elementów.

3.1.1 Dziedzina

Dziedzina to zbiór wszystkich obiektów pozostających w zainteresowaniu badacza, będących przedmiotem wnioskowania, oznaczana najczęściej przez X. Przykładowo mogą to być zbiory osób, transakcji, urządzeń, instytucji, itp.

3.1.2 Obserwacja

Każdy element dziedziny $x \in X$ nazywamy obserwacją. Obserwacją nazywać będziemy zarówno rekordy danych ze zbioru uczącego, jak i ze zbioru testowego.

3.1.3 Atrybuty obserwacji

Każdy obiekt z dziedziny $x \in X$ można opisać zestawem cech (atrybutów), które w notacji matematycznej oznaczymy przez $a : X \to A$, gdzie A jest przestrzenią wartości atrybutów. Każda obserwacja x posiadająca k cech da się wyrazić wektorowo jako $(a_1(x), a_2(x), ..., a_k(x))$. Dla większości algorytmów uczenia maszynowego wyróżnia się trzy typy atrybutów:

- *nominalne* - posiadające skończoną liczbę stanów, które posiadają porządku;
- *porządkowe* - posiadające skończoną liczbę stanów z zachowaniem porządku;
- *ciągłe* - przyjmujące wartości numeryczne.

Często jeden z atrybutów spełnia specjalną rolę, ponieważ stanowi realizację cechy, którą traktujemy jako wyjściową (ang. *target value attribute*). W tym przypadku powiemy o **nadzorowanym uczeniu maszynowym**. Jeśli zmiennej wyjściowej nie ma dziedzinie, to mówimy o **nienadzorowanym uczeniu maszynowym**.

3.1.4 Zbiór uczący

Zbiorem uczącym T (ang. *training set*) nazywamy podzbiór D dziedziny X (czyli $T \subseteq D \subseteq X$), gdzie zbiór D stanowi ogół dostępnych obserwacji z dziedziny X. Zbiór uczący zawiera informacje dotyczące badanego zjawiska, na podstawie których, dokonuje się doboru modelu, selekcji cech istotnych z punktu widzenia własności predykcyjnych lub jakości klasyfikacji, budowy modelu oraz optymalizacji jego parametrów. W przypadku uczenia z nauczycielem (nadzorowanego) zbiór T zawiera informację o wartościach atrybutów zmiennej wynikowej.

3.1.5 Zbiór testowy

Zbiór testowy T' (ang. *test set*) będący dopełnieniem zbioru uczącego do zbioru D, czyli $T' = D\ T$, stanowi zestaw danych służący do oceny poprawności modelu nadzorowanego. W przypadku metod nienadzorowanych raczej nie stosuje się zbiorów testowych.

3.1.6 Model

Model to narzędzie pozyskiwania wiedzy na podstawie zbioru uczącego. Nauczony model jest zbiorem reguł f, którego zadaniem jest oszacowanie wielkości wartości wynikowej lub odpowiednia klasyfikacja obiektów. W zadaniu grupowania obiektów (ang. *clustering task*), celem modelu jest podanie grup możliwie najbardziej jednorodnych przy zadanym zestawie zmiennych oraz ustalonej liczbie skupień (czasami wyznaczenie liczby skupień jest również częścią zadania stawianego przed modelem).

3.1.7 Jakość dopasowania modelu

Do oceny jakości dopasowania modelu wykorzystuje się, w zależności od zadania, wiele współczynników (np. dla zadań regresyjnych są to błąd średniokwadratowy - ang. *Mean Square Error*, a dla zadań klasyfikacyjnych - trafność - ang. *Accuracy*). Możemy mówić dwóch rodzajach dopasowania modeli:

- poziom dopasowania na zbiorze uczącym
- poziom dopasowania na zbiorze testowym (oczywiście z punktu widzenia utylitarności modelu ten współczynnik jest ważniejszy).

W sytuacji, w której model wykazuje dobre charakterystyki jakości dopasowania na zbiorze uczącym ale słabe na testowym, mówimy o zjawisku przeuczenia modelu (ang. *overfitting*). Oznacza to, że model wskazuje predykcję poprawnie jedynie dla zbioru treningowego ale ma słaba własności generalizacyjne nowe przypadki danych. Takie model nie przedstawiają znaczącej wartości w odkrywaniu wiedzy w sposób indukcyjny.

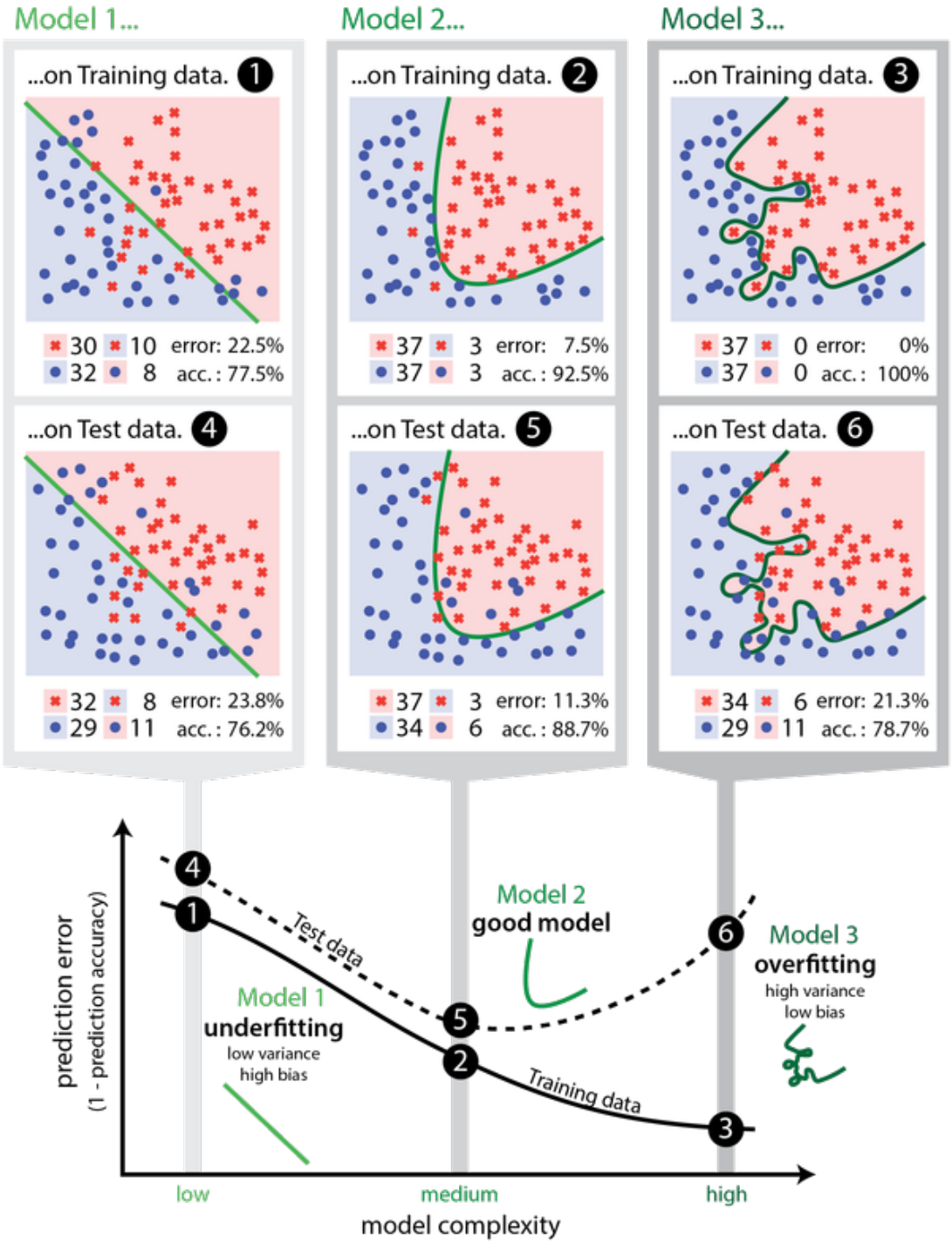
Z drugiej strony parametry dopasowania modelu mogą pokazywać słabe dopasowanie, zarówno na zbiorze uczącym, jak i testowym. Wówczas również model nie jest użyteczny w pozyskiwaniu wiedzy na temat badanego zjawiska, a sytuację taką nazywamy niedouczeniem (ang. *underfitting*).

3.2 Modele regresyjne

Jednym z rodzajów zadań bazującym na wnioskowaniu indukcyjnym jest model regresyjny. Należy on do grupy metod nadzorowanych, których celem jest oszacowanie wartości cechy wyjściowej (która jest ilościowa) na podstawie zestawu predyktorów, które mogą być ilościowe i jakościowe. Uczenie takich modeli odbywa się poprzez optymalizację funkcji celu (np. MSE) na podstawie zbioru uczącego.

3.3 Modele klasyfikacyjne

Podobnie jak modele regresyjne, modele klasyfikacyjne należą do grupy metod nadzorowanego uczenia maszynowego. Ich zadaniem jest właściwa klasyfikacja obiektów na podstawie wielkości predyktorów. Odpowiedzią modelu jest zawsze cecha typu jakościowego, natomiast predyktory mogą mieć dowolny typ. Wyróżnia się klasyfikację dwu i wielostanową. Lista modeli realizujących klasyfikację binarną jest nieco dłuższa niż w przypadku modeli z wielostanową cechą wynikową. Proces uczenia modelu klasyfikacyjnego również opiera się na optymalizacji funkcji celu. Tym razem są to zupełnie inne miary jakości dopasowania (np. trafność, czyli odsetek poprawnych klasyfikacji).



Rysunek 3.1: Przykłady niedoucznia (wykresy 1 i 4), poprawego modelu (2 i 5) i przeuczenia (3 i 6). Pierwszy wiersz wykresów pokazuje klasyfikację na podstawie modelu na zbiorze uczącym, a drugi na zbiorze testowym. Wykres na dole pokazuje związek pomiędzy złożonością modelu a wielkością błędu predykcji. *Źródło*: https://cambridgecoding.wordpress.com/2016/03/24/misleading-modelling-overfitting-cross-validation-and-the-bias-variance-trade-off/

3.4 Modele grupujące

Bardzo szeroką gamę modeli nienadzorowanych stanowią metody analizy skupień. Ich zadaniem jest grupowanie obiektów w możliwie najbardziej jednorodne grupy, na podstawie wartości atrybutów poddanych analizie. Ponieważ są to metody "bez nauczyciela", to ocena ich przydatności ma nieco inny charakter i choć istnieją różne wskaźniki jakości grupowania, to trudno tu o obiektywne wskazanie najlepszego rozwiązania.

Rozdział 4

Drzewa decyzyjne

Drzewo decyzyjne[1] jest strukturą hierarchiczną przedstawiającą model klasyfikacyjny lub regresyjny. Stosowane są szczególnie często wówczas, gdy funkcyjna postać związku pomiędzy predyktorami a zmienną wynikową jest nieznana lub ciężka do ustalenia. Każde drzewo decyzyjne składa się z korzenia (ang. *root*), węzłów (ang. *nodes*) i liści (ang. *leaves*). Korzeniem nazywamy początkowy węzeł drzewa, z którego poprzez podziały (ang. *splits*) powstają kolejne węzły potomne. Końcowe węzły, które nie podlegają podziałom nazywamy liśćmi, a linie łączące węzły nazywamy gałęziami (ang. *branches*).

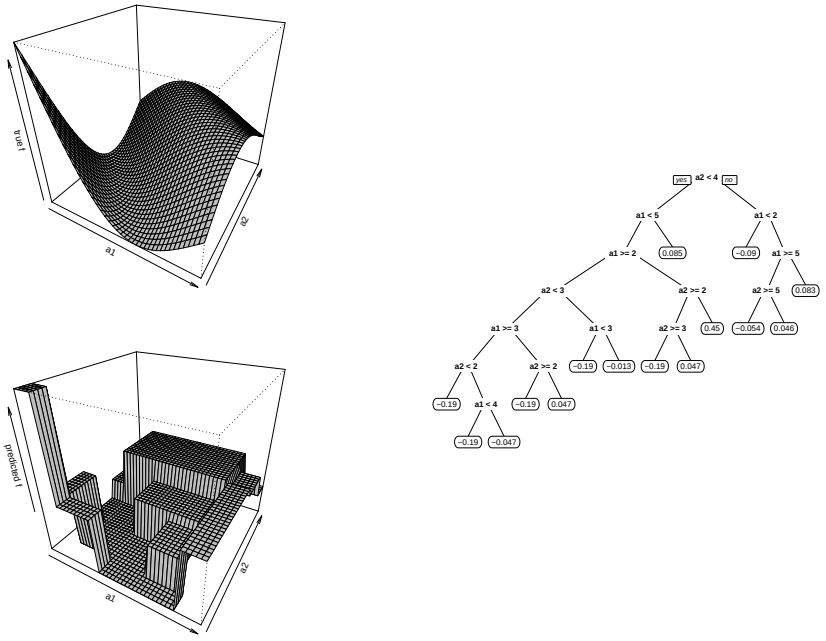
Jeśli drzewo służy do zadań klasyfikacyjnych, to liście zawierają informację o tym, która klasa w danym ciągu podziałów jest najbardziej prawdopodobna. Natomiast, jeśli drzewo jest regresyjne, to liście zawierają warunkowe miary tendencji centralnej (najczęściej średnią) wartości zmiennej wynikowej. Warunek stanowi szereg podziałów doprowadzający do danego węzła terminalnego (liścia). W obu przypadkach (klasyfikacji i regresji) drzewo "dąży" do takiego podziału by kolejne węzły, a co za tym idzie również liście, były ja najbardziej jednorodne ze względu na zmienną wynikową.

4.1 Węzły i gałęzie

Każdy podział rozdziela dziedzinę X na dwa lub więcej podobszarów dziedziny i wówczas każda obserwacja węzła nadrzędnego jest przyporządkowana węzłom potomnym. Każdy odchodzący węzeł potomny jest połączony gałęzią, która to wiąże się ściśle z możliwymi wynikami podziału. Każdy **n**-ty węzeł można opisać jako podzbiór dziedziny w następujący sposób

$$X_{\mathbf{n}} = \{x \in X | t_1(x) = r_1, t_2(x) = r_2, \dots, t_k(x) = r_k\}, \tag{4.1}$$

[1] wyglądem przypomina odwrócone drzewo, stąd nazwa



Rysunek 4.1: Przykład działania drzewa regresyjnego. Wykes w lewym górnym rogu pokazuje prawdziwą zależność, wyres po prawej stronie jest ilustracją drzewa decyzyjnego, a wykres w lewym dolnym rogu pokazuje dyskretyzację przestrzeni dokonaną przez drzewo, czyli sposób jego działania.

gdzie $t_1, t_2, \ldots, t_k$ są podziałami, które przeprowadzają x w obszary $r_1, r_2, \ldots, r_k$. Przez

$$S_{\mathbf{n},t=r} = \{x \in S | t(x) = r\} \tag{4.2}$$

rozumiemy, że dokonano takiego ciągu podziałów zbioru S, że jego wartości znalazły się w **n**-tym węźle.

4.2 Rodzaje reguł podziału

Najczęściej występujące reguły podziału w drzewach decyzyjnych są jednowymiarowe, czyli warunek podziału jest generowany na podstawie jednego atrybutu. Istnieją podziały wielowymiarowe ale ze względu na złożoność obliczeniową są rzadziej stosowane.

4.2.1 Podziały dla atrybutów ze skali nominalnej

Istnieją dwa typy reguł podziału dla skali nominalnej:

- oparte na wartości atrybutu (ang. *value based*) - wówczas funkcja testowa przyjmuje postać $t(x) = a(x)$, czyli podział generują wartości atrybutu;
- oparte na równości (ang. *equality based*) - gdzie funkcja testowa jest zdefiniowana jako

$$t(x) = \begin{cases} 1, & \text{gdy } a(x) = \nu \\ 0, & \text{w przeciwnym przypadku,} \end{cases} \tag{4.3}$$

  gdzie $\nu \in A$ i A jest zbiorem możliwych wartości a. W tym przypadku podział jest dychotomiczny, albo obiekt ma wartość atrybutu równą ν, albo go nie ma.

4.2.2 Podziały dla atrybutów ze skali ciągłej

Reguły podziału stosowane do skali ciągłej, to:

- oparta na nierównościach (ang. *inequality based*) - zdefiniowana jako

$$t(x) = \begin{cases} 1, & \text{gdy } a(x) \leq \nu \\ 0, & \text{w przeciwnym przypadku,} \end{cases} \tag{4.4}$$

  gdzie $\nu \in A$;
- przedziałowa (ang. *interval based*) - zdefiniowana jako

$$t(x) = \begin{cases} 1, & \text{gdy } a(x) \in I_1 \\ 2, & \text{gdy } a(x) \in I_2 \\ \vdots & \\ k, & \text{gdy } a(x) \in I_k \end{cases} \tag{4.5}$$

  gdzie $I_1, I_2, \ldots, I_k \subset A$ stanowią rozłączny podział (przedziałami) przeciwdziedziny A.

4.2.3 Podziały dla atrybutów ze skali porządkowej

Podziały te mogą wykorzystywać oba wcześniej wspomniane typy, w zależności od potrzeb.

4.3 Algorytm budowy drzewa

1. stwórz początkowy węzeł (korzeń) i oznacz go jako *otwarty*;
2. przypisz wszystkie możliwe rekordy do węzła początkowego;
3. **dopóki** istnieją otwarte węzły **wykonuj**:
   - wybierz węzeł $\mathbf{n}$, wyznacz potrzebne statystyki opisowe zmiennej zależnej dla tego węzła i przypisz wartość docelową;
   - **jeśli** kryterium zatrzymania podziału jest spełnione dla węzła n, **to** oznacz go za **zamknięty**;
   - **w przeciwnym przypadku** wybierz podział r elementów węzła $\mathbf{n}$, i dla każdego podzbioru podziału stwórz węzeł niższego rzędu (potomka) $\mathbf{n}_r$ oraz oznacz go jako *otwarty*;
   - następnie przypisz wszystkie przypadki generowane podziałem r do odpowiednich węzłów potomków $\mathbf{n}_r$;
   - oznacza węzeł $\mathbf{n}$ jako *zamknięty*.

Sposób przypisywania wartości docelowej wiąże się ściśle z rodzajem drzewa. W drzewach regresyjnych chodzi o wyliczenie średniej lub mediany dla obserwacji ujętych w danym węźle. Natomiast w przypadku drzewa klasyfikacyjnego, wyznacza się wartości prawdopodobieństw przynależności obserwacji znajdującej się w danym węźle do poszczególnych klas

$$\P(d|\mathbf{n}) = \P_{T_\mathbf{n}}(d) = \frac{|T^d_\mathbf{n}|}{|T_\mathbf{n}|}, \tag{4.6}$$

gdzie $T_\mathbf{n}$ oznaczają obserwacje zbioru uczącego znajdujące się w węźle $\mathbf{n}$, a $T^d_\mathbf{n}$ oznacza dodatkowo podzbiór zbioru uczącego w $\mathbf{n}$ węźle, które należą do klasy d. Oczywiście klasyfikacja na podstawie otrzymanych prawdopodobieństw w danym węźle jest dokonana przez wybór klasy charakteryzującej się najwyższym prawdopodobieństwem.

4.4 Kryteria zatrzymania

Kryterium zatrzymania jest warunkiem, który decyduje o tym, że dany węzeł uznajemy za zamknięty i nie dokonujemy dalszego jego podziału. Wyróżniamy następujące kryteria zatrzymania:

- jednorodność węzła - w przypadku drzewa klasyfikacyjnego może zdarzyć się sytuacja, że wszystkie obserwacje węzła będą pochodziły z jednej klasy. Wówczas nie ma sensu dokonywać dalszego podziału węzła;
- węzeł jest pusty - zbiór przypisanych obserwacji zbioru uczącego do $\mathbf{n}$-tego węzła jest pusty;

- brak reguł podziału - wszystkie reguły podziału zostały wykorzystane, zatem nie da się stworzyć potomnych węzłów, które charakteryzowałyby się większą homogenicznością;

Warunki ujęte w pierwszych dwóch kryteriach mogą być nieco złagodzone, poprzez zatrzymanie podziałów wówczas, gdy prawdopodobieństwo przynależenia do pewnej klasy przekroczy ustalony próg lub gdy liczebność węzła spadnie poniżej ustalonej wartości.

W literaturze tematu istnieje jeszcze jedno często stosowane kryterium zatrzymania oparte na wielkości drzewa. Węzeł potomny ustala się jako zamknięty, gdy długość ścieżki dojścia do nie go przekroczy ustaloną wartość.

4.5 Reguły podziału

Ważnym elementem algorytmu tworzenia drzewa regresyjnego jest *reguła podziału*. Dobierana jest w taki sposób aby zmaksymalizować zdolności generalizacyjne drzewa. Złożoność drzewa mierzona jest najczęściej przeciętną liczbą podziałów potrzebnych do dotarcia do liścia zaczynając od korzenia. Liście są najczęściej tworzone wówczas gdy dyspersja wartości wynikowej jest stosunkowo mała lub węzeł zawiera w miarę homogeniczne obserwacje ze względu na przynależność do klasy zmiennej wynikowej. W przypadku drzew regresyjnych zmienność na poziomie węzłów jest dobrą miarą służącą do definiowania podziału w węźle. I tak, jeśli pewien podział generuje nam stosunkowo małe dyspersje wartości docelowych w węzłach potomnych, to można ten podział uznać za właściwy. Jeśli T_n oznacza zbiór rekordów należących do węzła n, a $T_{n,t=r}$ są podzbiorami generowanymi przez podział r w węzłach potomnych dla n, to dyspersję wartości docelowej f będziemy oznaczali następująco

$$\mathrm{disp}_{T_{n,t=r}}(f). \tag{4.7}$$

Regułę podziału możemy określać poprzez minimalizację średniej ważonej dyspersji wartości docelowej następującej postaci

$$\mathrm{disp}_n(f|t) = \sum_{r \in R_t} \frac{|T_{n,t=r}|}{|T_n|} \mathrm{disp}_{T_{n,t=r}}(f), \tag{4.8}$$

gdzie $|\ |$ oznacza moc zbioru, a R_t zbiór wszystkich możliwych wartości reguły podziału. Czasami wygodniej będzie maksymalizować przyrost dyspersji (lub spadek)

$$\triangle \mathrm{disp}_n(f|t) = \mathrm{disp}_n(f) - \sum_{r \in R_t} \frac{|T_{n,t=r}|}{|T_n|} \mathrm{disp}_{T_{n,t=r}}(f). \tag{4.9}$$

Miarą heterogeniczności węzłów ze względu na zmienną wynikową (ang. *impurity*) w drzewach klasyfikacyjnych, która pozwala na tworzenie kolejnych podziałów węzła, są najczęściej wskaźnik Gini'ego i entropia (Breiman 1998).

Entropią podzbioru uczącego w węźle $\mathbf{n}$, wyznaczamy wg wzoru

$$E_{T_\mathbf{n}}(c|t) = \sum_{x \in R_t} \frac{|T_{\mathbf{n},t=r}|}{|T_\mathbf{n}|} E_{T_{\mathbf{n},t=r}}(c), \tag{4.10}$$

gdzie t jest podziałem (kandydatem), r potencjalnym wynikiem podziału t, c jest oznaczeniem klasy zmiennej wynikowej, a

$$E_{T_{\mathbf{n},t=r}}(c) = \sum_{d \in C} -¶_{T_{\mathbf{n},t=r}}(c = d) \log ¶_{T_{\mathbf{n},t=r}}(c = d), \tag{4.11}$$

przy czym

$$¶_{T_{\mathbf{n},t=r}}(c = d) = ¶_{T_\mathbf{n}}(c = d|t = r). \tag{4.12}$$

Podobnie definiuje się indeks Gini'ego

$$Gi_{T_\mathbf{n}}(c|t) = \sum_{x \in R_t} \frac{|T_{\mathbf{n},t=r}|}{|T_\mathbf{n}|} Gi_{T_{\mathbf{n},t=r}}(c), \tag{4.13}$$

gdzie

$$Gi_{T_{\mathbf{n},t=r}}(c) = \sum_{d \in C} ¶_{T_{\mathbf{n},t=r}}(c = d) \cdot (1 - ¶_{T_{\mathbf{n},t=r}}(c = d)) = 1 - \sum_{d \in C} ¶^2_{T_{\mathbf{n},t=r}}(c = d). \tag{4.14}$$

Dla tak zdefiniowanych miar "nieczystości" węzłów, podziału dokonujemy w taki sposób, aby zminimalizować współczynnik Gini'ego lub entropię. Im niższe miary nieczystości, tym bardziej obserwacje znajdujące się w węźle są monokulturą[2]. Nierzadko korzysta się również z współczynnika przyrostu informacji (ang. *information gain*)

$$\Delta E_{T_\mathbf{n}}(c|t) = E_{T_\mathbf{n}}(c) - E_{T_\mathbf{n}}(c|t). \tag{4.15}$$

Istnieje również jego odpowiednik dla indeksu Gini'ego. W obu przypadkach optymalnego podziału szukamy poprzez maksymalizację przyrostu informacji.

4.6 Przycinanie drzewa decyzyjnego

Uczenie drzewa decyzyjnego wiąże się z ryzykiem przeuczenia modelu (podobnie jak to się ma w przypadku innych modeli predykcyjnych). Wcześniej przytoczone reguły zatrzymania (np. głębokość drzewa czy zatrzymanie przy osiągnięciu jednorodności na zadanym poziomie) pomagają kontrolować poziom generalizacji drzewa ale czasami będzie dodatkowo potrzebne przycięcie drzewa, czyli usunięcie pewnych podziałów, a co za tym idzie, również liści (węzłów).

[2] prawie wszystkie są w jednej klasie

4.6.1 Przycinanie redukujące błąd

Jedną ze strategii przycinania drzewa jest przycinanie redukujące błąd (ang. *reduced error pruning*). Polega ono na porównaniu błędów (najczęściej używana jest miara odsetka błędnych klasyfikacji lub MSE) liścia $\mathbf{l}$ i węzła do którego drzewo przycinamy $\mathbf{n}$ na całkiem nowym zbiorze uczącym R. Niech $e_R(\mathbf{l})$ i $e_R(\mathbf{n})$ oznaczają odpowiednio błędy na zbiorze R liścia i węzła. Przez błąd węzła rozumiemy błąd pod-drzewa o korzeniu $\mathbf{n}$. Wówczas jeśli zachodzi warunek

$$e_R(\mathbf{l}) \leq e_R(\mathbf{n}), \tag{4.16}$$

to zaleca się zastąpić węzeł $\mathbf{n}$ liściem $\mathbf{l}$.

4.6.2 Przycinanie minimalizujące błąd

Przycinanie minimalizujące błąd opiera się na spostrzeżeniu, że błąd drzewa przyciętego charakteryzuje się zbyt pesymistyczną oceną i dlatego wymaga korekty. Węzeł drzewa klasyfikacyjnego $\mathbf{n}$ zastępujemy liściem $\mathbf{l}$, jeśli

$$\hat{e}_T(\mathbf{l}) \leq \hat{e}_T(\mathbf{n}), \tag{4.17}$$

gdzie

$$\hat{e}_T(\mathbf{n}) = \sum_{\mathbf{n}' \in N(\mathbf{n})} \frac{|T_{\mathbf{n}'}|}{|T_{\mathbf{n}}|} \hat{e}_T(\mathbf{n}'), \tag{4.18}$$

a $N(\mathbf{n})$ jest zbiorem wszystkich możliwych węzłów potomnych węzła $\mathbf{n}$ i

$$\hat{e}_T(\mathbf{l}) = 1 - \frac{|\{x \in T_{\mathbf{l}} | c(x) = d_{\mathbf{l}}\}| + mp}{|T_{\mathbf{l}}| + m}, \tag{4.19}$$

gdzie p jest prawdopodobieństwem przynależności do klasy $d_{\mathbf{l}}$ ustalona na podstawie zewnętrznej wiedzy (gdy jej nie posiadamy przyjmujemy $p = 1/|C|$).

W przypadku drzewa regresyjnego znajdujemy wiele analogii, ponieważ jeśli dla pewnego zbioru rekordów T spełniony jest warunek

$$\mathrm{mse}_T(\mathbf{l}) \leq \mathrm{mse}_T(\mathbf{n}), \tag{4.20}$$

gdzie $\mathbf{l}$ i $\mathbf{n}$ oznaczają odpowiednio liść i węzeł, to wówczas zastępujemy węzeł $\mathbf{n}$ przez liść $\mathbf{l}$.

Estymatory wyznaczone na podstawie niewielkiej próby, mogą być obarczone znaczącym błędem. Wyliczanie błędu średnio-kwadratowego dla podzbioru nowych wartości może się charakteryzować takim obciążeniem. Dlatego stosuje się statystyki opisowe z poprawką, której pochodzenie może mieć trzy źródła: wiedza merytoryczna na temat szukanej wartości, założeń modelu lub na podstawie wyliczeń opartych o cały zbiór wartości.

Skorygowany estymator błędu średnio-kwadratowego ma następującą postać

$$\widehat{\mathrm{mse}}_T(\mathbf{l}) = \frac{\sum_{x \in T} (f(x) - m_{\mathbf{l},m,m_0}(f))^2 + mS_0^2}{|T_{\mathbf{l}}| + m}, \tag{4.21}$$

gdzie

$$m_{\mathbf{l},m,m_0}(f) = \frac{\sum_{x \in T_\mathbf{l}} f(x) + m m_0}{|T_\mathbf{l}| + m}, \tag{4.22}$$

a m_0 i S_0^2 są średnią i wariancją wyznaczonymi na całej próbie uczącej. Błąd średnio-kwadratowy węzła $\mathbf{n}$ ma postać

$$\widehat{\mathrm{mse}}_T(\mathbf{n}) = \sum_{\mathbf{n}' \in N(\mathbf{n})} \frac{|T_{\mathbf{n}'}|}{|T_\mathbf{n}|} \widehat{\mathrm{mse}}_T(\mathbf{n}'). \tag{4.23}$$

Wówczas kryterium podcięcia można zapisać w następujący sposób

$$\widehat{\mathrm{mse}}_T(\mathbf{l}) \leq \widehat{\mathrm{mse}}_T(\mathbf{n}) \tag{4.24}$$

4.6.3 Przycinanie ze względu na współczynnik złożoności drzewa

Przycinanie ze względu na współczynnik złożoności drzewa (ang. *cost-complexity pruning*) polega na wprowadzeniu "kary" za zwiększoną złożoność drzewa. Drzewa klasyfikacyjne przycinamy gdy spełniony jest warunek

$$e_T(\mathbf{l}) \leq e_T(\mathbf{n}) + \alpha C(\mathbf{n}), \tag{4.25}$$

gdzie $C(\mathbf{n})$ oznacza złożoność drzewa mierzoną liczbą liści, a α parametrem wagi kary za złożoność drzewa.

Wspomniane kryterium przycięcia dla drzew regresyjnych bazuje na względnym błędzie średnio-kwadratowym (ang. *relative square error*), czyli

$$\widehat{\mathrm{rse}}_T(\mathbf{n}) = \frac{|T| \widehat{\mathrm{mse}}_T(\mathbf{n})}{(|T|-1) S_T^2(f)}, \tag{4.26}$$

gdzie T oznacza podzbiór X, S_T^2 wariancję na zbiorze T. Wówczas kryterium podcięcia wygląda następująco

$$\widehat{\mathrm{rse}}_T(\mathbf{l}) \leq \widehat{\mathrm{rse}}_T(\mathbf{n}) + \alpha C(\mathbf{n}). \tag{4.27}$$

4.7 Obsługa braków danych

Drzewa decyzyjne wyjątkowo dobrze radzą sobie z obsługa zbiorów z brakami. Stosowane są głównie dwie strategie:

- udziałów obserwacji (ang. *fractional instances*) - rozważane są wszystkie możliwe podziały dla brakującej obserwacji i przypisywana jest im odpowiednia waga lub prawdopodobieństwo, w oparciu o zaobserwowany rozkład znanych obserwacji. Te same wagi są stosowane do predykcji wartości na podstawie drzewa z brakami danych.

- podziałów zastępczych (ang. *surrogate splits*) - jeśli wynik podziału nie może być ustalony dla obserwacji z brakami, to używany jest podział zastępczy (pierwszy), jeśli i ten nie może zostać ustalony, to stosuje się kolejny. Kolejne podziały zastępcze są generowane tak, aby wynik podziału możliwie najbardziej przypominał podział właściwy.

4.8 Zalety i wady

4.8.1 Zalety

- łatwe w interpretacji;
- nie wymagają żmudnego przygotowania danych (brak standaryzacji, wprowadzania zmiennych binarnych, dopuszcza występowanie braków danych);
- działa na obu typach zmiennych - jakościowych i ilościowych;
- dopuszcza nieliniowość związku między zmienną wynikową a predyktorami;
- odporny na odstępstwa od założeń;
- pozwala na obsługę dużych zbiorów danych.

4.8.2 Wady

- brak jawnej postaci zależności;
- zależność struktury drzewa od użytego algorytmu;
- przegrywa jakością predykcji z innymi metodami nadzorowanego uczenia maszynowego.

Przykład 4.1. Przykładem zastosowania drzew decyzyjnych będzie klasyfikacja irysów na podstawie długości i szerokości kielicha i płatka.

Przykładem zastosowania drzew decyzyjnych będzie klasyfikacja irysów na podstawie długości i szerokości kielicha i płatka.

```
library(tidyverse)
library(rpart) # pakiet do tworzenia drzew typu CART
library(rpart.plot) # pakiet do rysowania drzew
```

Każde zadanie ucznia maszynowego zaczynamy od czyszczenia danych i odpowiedniego ich przygotowania ale w tym przypadku skupimy się jedynie na budowie, optymalizacji i ewaluacji modelu.

Podział zbioru na próbę uczącą i testową

```
set.seed(2019)
dt.train <- iris %>%
    sample_frac(size = 0.7)
dt.test <- setdiff(iris, dt.train)
str(dt.train)
```

```
## 'data.frame':    105 obs. of  5 variables:
```

```
##  $ Sepal.Length: num  4.8 6.7 6.2 5.4 7.7 5 5.7 5 6.3 6.8 ...
##  $ Sepal.Width : num  3.4 3.3 2.2 3.9 2.6 2 3.8 3 3.4 3 ...
##  $ Petal.Length: num  1.9 5.7 4.5 1.3 6.9 3.5 1.7 1.6 5.6 5.5 ...
##  $ Petal.Width : num  0.2 2.1 1.5 0.4 2.3 1 0.3 0.2 2.4 2.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 3 2 1 3 2 1 1 3 3 .
```

```
str(dt.test)
```

```
## 'data.frame':    45 obs. of  5 variables:
##  $ Sepal.Length: num  4.4 5.4 4.8 4.3 5.7 5.1 5.2 5.2 5.2 4.9 ...
##  $ Sepal.Width : num  2.9 3.7 3 3 4.4 3.8 3.5 3.4 4.1 3.1 ...
##  $ Petal.Length: num  1.4 1.5 1.4 1.1 1.5 1.5 1.5 1.4 1.5 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.1 0.1 0.4 0.3 0.2 0.2 0.1 0.2 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 .
```

Budowa drzewa

Budowy drzewa dokonujemy za pomocą funkcji `rpart` pakietu **rpart** (Therneau and Atkinson 2018) stosując zapis formuły zależności. Drzewo zostanie zbudowane z uwzględnieniem kilku kryteriów zatrzymania:

- minimalna liczebność węzła, który może zostać podzielony to 10 - ze względu na małą liczebność zbioru uczącego;
- minimalna liczebność liścia to 5 - aby nie dopuścić do przeuczenia modelu;
- maksymalna głębokość drzewa to 4 - aby nie dopuścić do przeuczenia modelu.

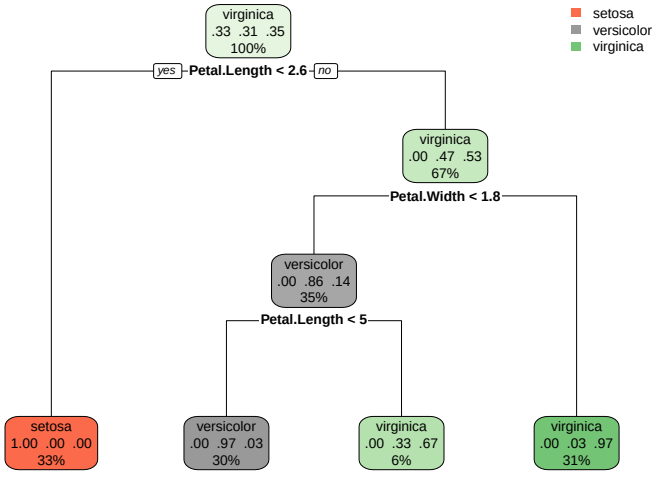
```
mod.rpart <- rpart(Species~., data = dt.train,
                   control = rpart.control(minsplit = 10,
                                           minbucket = 5,
                                           maxdepth = 4))
summary(mod.rpart)
```

```
## Call:
## rpart(formula = Species ~ ., data = dt.train, control = rpart.control(minsplit = 10
##     minbucket = 5, maxdepth = 4))
##   n= 105
##
##           CP nsplit  rel error    xerror       xstd
## 1 0.51470588      0 1.00000000 1.1176471 0.06737554
## 2 0.39705882      1 0.48529412 0.4852941 0.06995514
## 3 0.02941176      2 0.08823529 0.1617647 0.04614841
## 4 0.01000000      3 0.05882353 0.1617647 0.04614841
##
## Variable importance
##  Petal.Width Petal.Length Sepal.Length  Sepal.Width
##           33           32           20           15
##
## Node number 1: 105 observations,    complexity param=0.5147059
```

```
##   predicted class=virginica   expected loss=0.647619  P(node) =1
##     class counts:    35    33    37
##    probabilities: 0.333 0.314 0.352
##   left son=2 (35 obs) right son=3 (70 obs)
##   Primary splits:
##       Petal.Length < 2.6  to the left,  improve=35.03810, (0 missing)
##       Petal.Width  < 0.8  to the left,  improve=35.03810, (0 missing)
##       Sepal.Length < 5.45 to the left,  improve=25.60255, (0 missing)
##       Sepal.Width  < 3.35 to the right, improve=14.70881, (0 missing)
##   Surrogate splits:
##       Petal.Width  < 0.8  to the left,  agree=1.000, adj=1.000, (0 split)
##       Sepal.Length < 5.45 to the left,  agree=0.933, adj=0.800, (0 split)
##       Sepal.Width  < 3.35 to the right, agree=0.848, adj=0.543, (0 split)
##
## Node number 2: 35 observations
##   predicted class=setosa      expected loss=0  P(node) =0.3333333
##     class counts:    35     0     0
##    probabilities: 1.000 0.000 0.000
##
## Node number 3: 70 observations,    complexity param=0.3970588
##   predicted class=virginica   expected loss=0.4714286  P(node) =0.6666667
##     class counts:     0    33    37
##    probabilities: 0.000 0.471 0.529
##   left son=6 (37 obs) right son=7 (33 obs)
##   Primary splits:
##       Petal.Width  < 1.75 to the left,  improve=24.297670, (0 missing)
##       Petal.Length < 4.75 to the left,  improve=24.174190, (0 missing)
##       Sepal.Length < 5.75 to the left,  improve= 4.483555, (0 missing)
##       Sepal.Width  < 2.55 to the left,  improve= 3.793760, (0 missing)
##   Surrogate splits:
##       Petal.Length < 4.75 to the left,  agree=0.886, adj=0.758, (0 split)
##       Sepal.Length < 6.15 to the left,  agree=0.671, adj=0.303, (0 split)
##       Sepal.Width  < 2.65 to the left,  agree=0.671, adj=0.303, (0 split)
##
## Node number 6: 37 observations,    complexity param=0.02941176
##   predicted class=versicolor  expected loss=0.1351351  P(node) =0.352381
##     class counts:     0    32     5
##    probabilities: 0.000 0.865 0.135
##   left son=12 (31 obs) right son=13 (6 obs)
##   Primary splits:
##       Petal.Length < 4.95 to the left,  improve=4.0464980, (0 missing)
##       Petal.Width  < 1.35 to the left,  improve=1.0296010, (0 missing)
##       Sepal.Width  < 3.05 to the right, improve=0.2615519, (0 missing)
##       Sepal.Length < 5.95 to the left,  improve=0.1828101, (0 missing)
##
## Node number 7: 33 observations
```

```
##   predicted class=virginica   expected loss=0.03030303  P(node) =0.3142857
##     class counts:     0     1    32
##    probabilities: 0.000 0.030 0.970
##
## Node number 12: 31 observations
##   predicted class=versicolor  expected loss=0.03225806  P(node) =0.2952381
##     class counts:     0    30     1
##    probabilities: 0.000 0.968 0.032
##
## Node number 13: 6 observations
##   predicted class=virginica   expected loss=0.3333333  P(node) =0.05714286
##     class counts:     0     2     4
##    probabilities: 0.000 0.333 0.667
```

```
rpart.plot(mod.rpart)
```



Rysunek 4.2: Obraz drzewa klasyfikacyjnego.

Powyższy wykres przedstawia strukturę drzewa klasyfikacyjnego. Kolorami są oznaczone klasy, które w danym węźle dominują. Nasycenie barwy decyduje o sile tej dominacji. W każdym węźle podana jest klasa, do której najprawdopodobniej należą jego obserwacje. Ponadto podane są proporcje przynależności do klas zmiennej wynikowej oraz procent obserwacji zbioru uczącego należących do danego węzła. Pod każdym węzłem podana jest reguła podziału.

Przycinanie drzewa

Zanim przystąpimy do przycinania drzewa należy sprawdzić, jakie są zdolności generalizacyjne modelu. Oceny tej dokonujemy najczęściej sprawdzając macierz klasyfikacji.

```
pred.prob <- predict(mod.rpart,
                     newdata = dt.test)
pred.prob[10:20,]
```

```
##    setosa versicolor  virginica
## 10      1  0.0000000 0.00000000
## 11      1  0.0000000 0.00000000
## 12      1  0.0000000 0.00000000
## 13      1  0.0000000 0.00000000
## 14      1  0.0000000 0.00000000
## 15      1  0.0000000 0.00000000
## 16      0  0.9677419 0.03225806
## 17      0  0.9677419 0.03225806
## 18      0  0.9677419 0.03225806
## 19      0  0.9677419 0.03225806
## 20      0  0.9677419 0.03225806
```

```
pred.class <- predict(mod.rpart,
                      newdata = dt.test,
                      type = "class")
pred.class
```

```
##          1          2          3          4          5          6          7
##     setosa     setosa     setosa     setosa     setosa     setosa     setosa
##          8          9         10         11         12         13         14
##     setosa     setosa     setosa     setosa     setosa     setosa     setosa
##         15         16         17         18         19         20         21
##     setosa versicolor versicolor versicolor versicolor versicolor versicolor
##         22         23         24         25         26         27         28
## versicolor versicolor versicolor versicolor versicolor versicolor versicolor
##         29         30         31         32         33         34         35
## versicolor versicolor versicolor versicolor  virginica  virginica  virginica
##         36         37         38         39         40         41         42
##  virginica  virginica  virginica  virginica  virginica  virginica  virginica
##         43         44         45
##  virginica  virginica  virginica
## Levels: setosa versicolor virginica
```

```
tab <- table(predykcja = pred.class, obserwacja = dt.test$Species)
tab
```

```
##             obserwacja
```

```
## predykcja    setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         17         0
##   virginica       0          0        13
```

Jak widać z powyższej tabeli, model całkiem dobrze radzi sobie z poprawną klasyfikacją obserwacji do odpowiednich kategorii.

W dalszej kolejności sprawdzimy, czy nie jest konieczne przycięcie drzewa. Jednym z kryteriów przycinania drzewa jest przycinanie ze względu na złożoność drzewa. W tym przypadku jest wyrażony parametrem `cp`. Istnieje powszechnie stosowana reguła jednego odchylenia standardowego, która mówi, że drzewo należy przyciąć wówczas, gdy błąd oszacowany na podstawie sprawdzianu krzyżowego (`xerror`), pierwszy raz zejdzie poniżej poziomu wyznaczonego przez najniższą wartość błędu powiększonego o odchylenie standardowe tego błędu (`xstd`). Na podstawie poniższej tabeli można ustalić, że poziomem odcięcia jest wartość $0.16176 + 0.046148 = 0.207908$. Pierwszy raz błąd przyjmuje wartość mniejszą od 0.16176 po drugim podziale (`nsplit=2`). Temu poziomowi odpowiada `cp` o wartości 0.029412 i to jest złożoność drzewa, którą powinniśmy przyjąć do przycięcia drzewa.

```
printcp(mod.rpart)
```

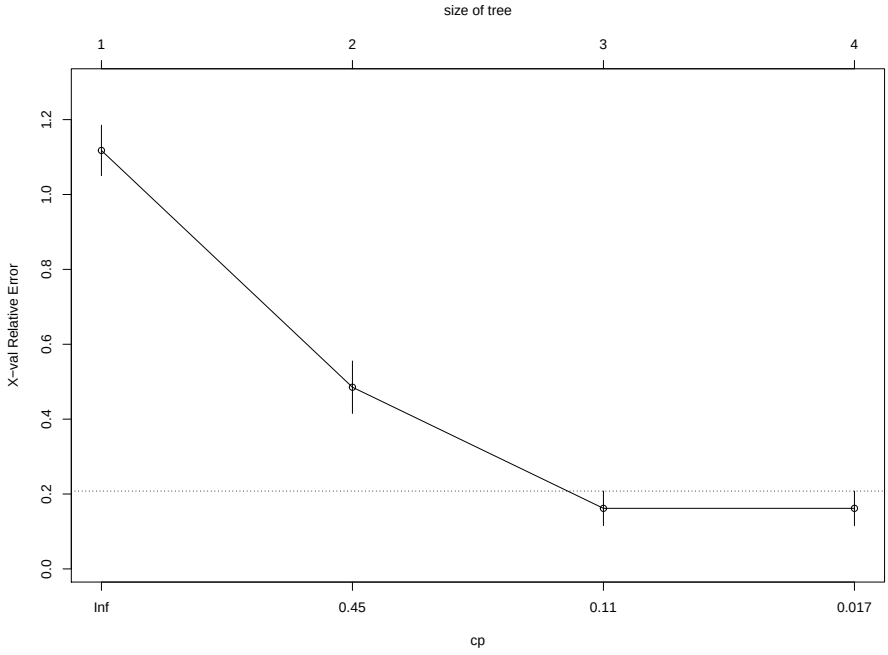
```
##
## Classification tree:
## rpart(formula = Species ~ ., data = dt.train, control = rpart.control(minsplit = 10
##     minbucket = 5, maxdepth = 4))
##
## Variables actually used in tree construction:
## [1] Petal.Length Petal.Width
##
## Root node error: 68/105 = 0.64762
##
## n= 105
##
##         CP nsplit rel error  xerror     xstd
## 1 0.514706      0  1.000000 1.11765 0.067376
## 2 0.397059      1  0.485294 0.48529 0.069955
## 3 0.029412      2  0.088235 0.16176 0.046148
## 4 0.010000      3  0.058824 0.16176 0.046148
```

```
plotcp(mod.rpart)
```

Przycięte drzewo wygląda następująco:

```
mod.rpart2 <- prune(mod.rpart, cp = 0.029412)
summary(mod.rpart2)
```

```
## Call:
```

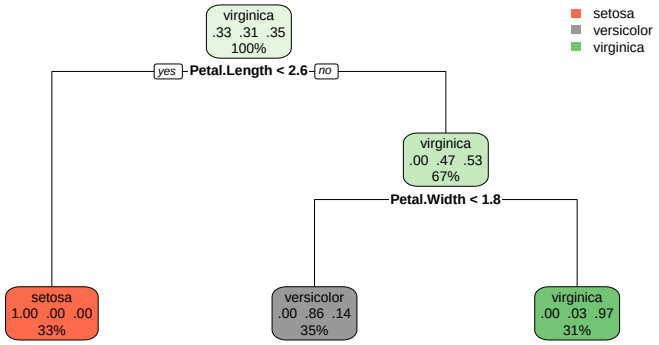


Rysunek 4.3: Na wykresie błędów punkt odcięcia zaznaczony jest linią przerywaną

```
## rpart(formula = Species ~ ., data = dt.train, control = rpart.control(minsplit = 10
##     minbucket = 5, maxdepth = 4))
##   n= 105
##
##          CP nsplit  rel error    xerror       xstd
## 1 0.5147059      0 1.00000000 1.1176471 0.06737554
## 2 0.3970588      1 0.48529412 0.4852941 0.06995514
## 3 0.0294120      2 0.08823529 0.1617647 0.04614841
##
## Variable importance
##  Petal.Width Petal.Length Sepal.Length  Sepal.Width
##           34           31           20           15
##
## Node number 1: 105 observations,    complexity param=0.5147059
##   predicted class=virginica   expected loss=0.647619  P(node) =1
##     class counts:    35    33    37
##    probabilities: 0.333 0.314 0.352
##   left son=2 (35 obs) right son=3 (70 obs)
##   Primary splits:
##       Petal.Length < 2.6  to the left,  improve=35.03810, (0 missing)
##       Petal.Width  < 0.8  to the left,  improve=35.03810, (0 missing)
##       Sepal.Length < 5.45 to the left,  improve=25.60255, (0 missing)
##       Sepal.Width  < 3.35 to the right, improve=14.70881, (0 missing)
##   Surrogate splits:
##       Petal.Width  < 0.8  to the left,  agree=1.000, adj=1.000, (0 split)
##       Sepal.Length < 5.45 to the left,  agree=0.933, adj=0.800, (0 split)
##       Sepal.Width  < 3.35 to the right, agree=0.848, adj=0.543, (0 split)
##
## Node number 2: 35 observations
##   predicted class=setosa      expected loss=0  P(node) =0.3333333
##     class counts:    35     0     0
##    probabilities: 1.000 0.000 0.000
##
## Node number 3: 70 observations,    complexity param=0.3970588
##   predicted class=virginica   expected loss=0.4714286  P(node) =0.6666667
##     class counts:     0    33    37
##    probabilities: 0.000 0.471 0.529
##   left son=6 (37 obs) right son=7 (33 obs)
##   Primary splits:
##       Petal.Width  < 1.75 to the left,  improve=24.297670, (0 missing)
##       Petal.Length < 4.75 to the left,  improve=24.174190, (0 missing)
##       Sepal.Length < 5.75 to the left,  improve= 4.483555, (0 missing)
##       Sepal.Width  < 2.55 to the left,  improve= 3.793760, (0 missing)
##   Surrogate splits:
##       Petal.Length < 4.75 to the left,  agree=0.886, adj=0.758, (0 split)
##       Sepal.Length < 6.15 to the left,  agree=0.671, adj=0.303, (0 split)
```

```
##       Sepal.Width  < 2.65 to the left,  agree=0.671, adj=0.303, (0 split)
##
## Node number 6: 37 observations
##   predicted class=versicolor  expected loss=0.1351351  P(node) =0.352381
##     class counts:     0    32     5
##    probabilities: 0.000 0.865 0.135
##
## Node number 7: 33 observations
##   predicted class=virginica   expected loss=0.03030303  P(node) =0.3142857
##     class counts:     0     1    32
##    probabilities: 0.000 0.030 0.970
```

```
rpart.plot(mod.rpart2)
```



Rysunek 4.4: Drzewo klasyfikacyjne po przycięciu

Ocena dopasowania modelu

Na koniec budowy modelu należy sprawdzić jego jakość na zbiorze testowym.

```
pred.class2 <- predict(mod.rpart2,
                       newdata = dt.test,
                       type = "class")
tab2 <- table(predykcja = pred.class2, obserwacja = dt.test$Species)
tab2
```

```
##             obserwacja
## predykcja    setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         17         0
##   virginica       0          0        13
```

Mimo przycięcia drzewa, klasyfikacja pozostaje na niezmienionym poziomie. Odsetek poprawnych klasyfikacji możemy oszacować za pomocą

```
round(sum(diag(tab2))/sum(tab2)*100,1)
```

```
## [1] 100
```

4.9 Inne algorytmy budowy drzew decyzyjnych implementowane w R

Oprócz najbardziej znanego algorytmu CART implementowanego w postaci funkcji pakietu **rpart**, istnieją również inne algorytmy, które znalazły swoje implementacje w R. Są to:

- *CHAID*[3] - algorytm przeznaczony do budowy drzew klasyfikacyjnych, gdzie zarówno zmienna wynikowa, jak i zmienne niezależne muszą być ze skali jakościowej. Główną różnicą w stosunku do drzew typu CART jest sposób budowy podziałów, oparty na teście niezależności χ^2 Pearsona. Wyboru reguły podziału dokonuje się poprzez testowanie niezależności zmiennej niezależnej z predyktorami. Reguła o największej wartości statystyki χ^2 jest stosowana w pierwszej kolejności. Implementacja tego algorytmu znajduje się w pakiecie **CHAID**[4] (funkcja do tworzenia drzewa o tej samej nazwie `chaid`) (Team 2015).
- *Ctree*[5] - algorytm zbliżony zasadą działania do CHAID, ponieważ również wykorzystuje testowanie do wyboru reguły podziału. Różni się jednak tym, że może być stosowany do zmiennych dowolnego typu oraz tym, że może być zarówno drzewem klasyfikacyjnym jak i regresyjnym. Implementację R-ową można znaleźć w pakietach **party** (Hothorn, Hornik, and Zeileis 2006) lub **partykit** (Hothorn and Zeileis 2015) - funkcją do tworzenia modelu jest `ctree`.
- *C4.5* - algorytm stworzony przez Quinlan (1993) w oparciu, o również jego autorstwa, algorytm ID3. Służy jedynie do zadań klasyfikacyjnych. W dużym uproszczeniu, dobór reguł podziału odbywa się na podstawie przyrostu informacji (patrz Reguły podziału). W przeciwieństwie do pierwotnego algorytmu ID3, C4.5 nie raczej nie przeucza drzew. Implementacja R-owa znajduje się w pakiecie **RWeka** (Hornik, Buchta, and Zeileis 2009) - funkcja do budowy drzewa to `J48`.

[3] Chi-square automatic interaction detection
[4] brak w oficjalnej dystrybucji CRAN
[5] Conditional Inference Trees

- *C5.0* - kolejny algorytm autorstwa Kuhn and Quinlan (2018) jest usprawnieniem algorytmu C4.5, generującym mniejsze drzewa automatycznie przycinane na podstawie złożoności drzewa. Służy jedynie do zadań klasyfikacyjnych. Jest szybszy od poprzednika i pozwala na zastosowanie metody *boosting*[6]. Implementacja R-owa znajduje się w pakiecie *C50*, a funkcja do budowy drzewa to `C5.0`.

Przykład 4.2. W celu porównania wyników klasyfikacji na podstawie drzew decyzyjnych o różnych algorytmach, zostaną nauczone modele w oparciu o funkcje `ctree`, `J48` i `C5.0` dla tego samego zestawu danych co w przykładzie wcześniejszym **??**.

- **Drzewo `ctree`**

Na początek ustalamy parametry ograniczające rozrost drzewa podobne jak w poprzednim przykładzie.

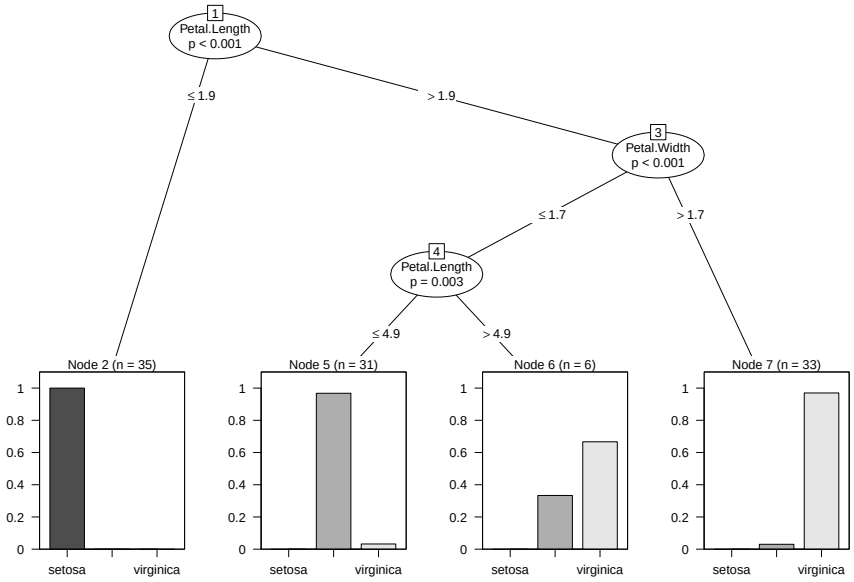
```
library(partykit)
tree2 <- ctree(Species~., data = dt.train,
               control = ctree_control(minsplit = 10,
                                       minbucket = 5,
                                       maxdepth = 4))
tree2
```

```
##
## Model formula:
## Species ~ Sepal.Length + Sepal.Width + Petal.Length + Petal.Width
##
## Fitted party:
## [1] root
## |   [2] Petal.Length <= 1.9: setosa (n = 35, err = 0.0%)
## |   [3] Petal.Length > 1.9
## |   |   [4] Petal.Width <= 1.7
## |   |   |   [5] Petal.Length <= 4.9: versicolor (n = 31, err = 3.2%)
## |   |   |   [6] Petal.Length > 4.9: virginica (n = 6, err = 33.3%)
## |   |   [7] Petal.Width > 1.7: virginica (n = 33, err = 3.0%)
##
## Number of inner nodes:    3
## Number of terminal nodes: 4
```

```
plot(tree2)
```

Wydaje się, że drzewo nie jest optymalne, ponieważ w węźle 6 obserwacje z grup `versicolor` i `virginica` są nieco pomieszane. Ostateczne oceny dokonujemy na podstawie próby testowej.

[6] budowa klasyfikatora w oparciu o proces iteracyjny, w którym kolejne w kolejnych iteracjach budowane są proste drzewa i przypisywane są im wagi - im gorszy klasyfikator, tym większa waga - po to aby nauczyć drzewo klasyfikować "trudne" przypadki



Rysunek 4.5: Wykres drzewa decyzyjnego zbudowanego metodą ctree

```
pred2 <- predict(tree2, newdata = dt.test)
tab <- table(predykcja = pred2, obserwacja = dt.test$Species)
tab
```

```
##             obserwacja
## predykcja    setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         17         0
##   virginica       0          0        13
```

Dopiero ocena jakości klasyfikacji na podstawie próby testowej pokazuje, że model zbudowany za pomocą `ctree` daje podobną precyzję jak `rpart` przycięty.

- **Drzewo `J48`**

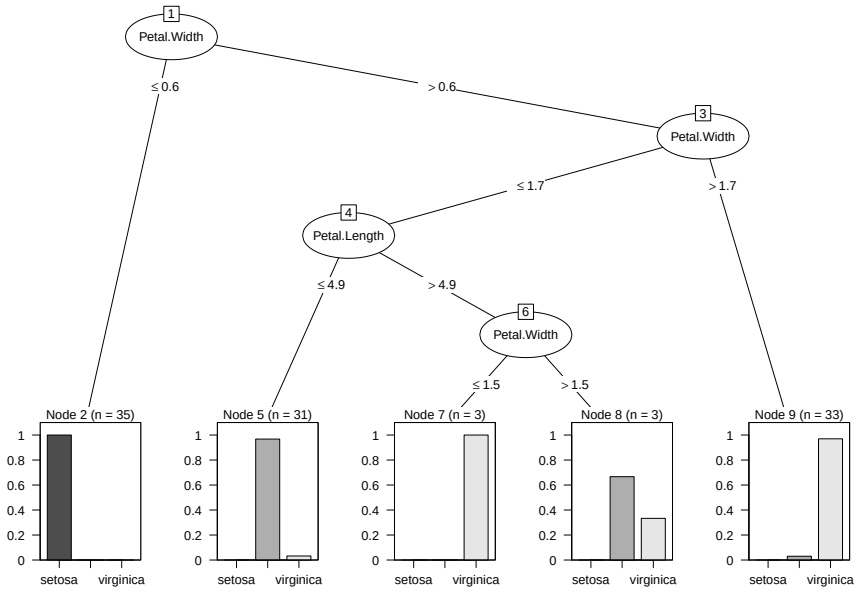
W tym przypadku model sam poszukuje optymalnego rozwiązania przycinając się automatycznie.

```
library(RWeka)
tree3 <- J48(Species~., data = dt.train)
tree3
```

```
## J48 pruned tree
## ------------------
##
```

```
## Petal.Width <= 0.6: setosa (35.0)
## Petal.Width > 0.6
## |   Petal.Width <= 1.7
## |   |   Petal.Length <= 4.9: versicolor (31.0/1.0)
## |   |   Petal.Length > 4.9
## |   |   |   Petal.Width <= 1.5: virginica (3.0)
## |   |   |   Petal.Width > 1.5: versicolor (3.0/1.0)
## |   Petal.Width > 1.7: virginica (33.0/1.0)
##
## Number of Leaves  :  5
##
## Size of the tree :   9
```

```
plot(tree3)
```



Rysunek 4.6: Wykres drzewa decyzyjnego zbudowanego metodą J48

Drzewo jest nieco bardziej rozbudowane niż `tree2` i `mod.rpart2`.

```
summary(tree3)
```

```
##
## === Summary ===
##
## Correctly Classified Instances         102               97.1429 %
## Incorrectly Classified Instances         3                2.8571 %
## Kappa statistic                          0.9571
```

```
## Mean absolute error                       0.0331
## Root mean squared error                   0.1286
## Relative absolute error                   7.4482 %
## Root relative squared error              27.2918 %
## Total Number of Instances               105
##
## === Confusion Matrix ===
##
##   a  b  c   <-- classified as
##  35  0  0 |  a = setosa
##   0 32  1 |  b = versicolor
##   0  2 35 |  c = virginica
```

Podsumowanie dopasowania drzewa na próbie uczącej jest bardzo dobre, bo poprawnych klasyfikacji jest ponad 97%. Oceny dopasowania i tak dokonujemy na zbiorze testowym.

```
pred3 <- predict(tree3, newdata = dt.test)
tab <- table(predykcja = pred3, obserwacja = dt.test$Species)
tab
```

```
##             obserwacja
## predykcja    setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         17         0
##   virginica       0          0        13
```

Otrzymujemy identyczną macierz klasyfikacji jak w poprzednich przypadkach.

- **Drzewo `C50`**

Tym razem również nie trzeba ustawiać parametrów drzewa, ponieważ algorytm działa tak aby zapobiec rozrostowi drzewa przy jednoczesnej wysokiej poprawności klasyfikacji.

```
library(C50)
tree4 <- C5.0(Species~., data = dt.train)
summary(tree4)
```

```
##
## Call:
## C5.0.formula(formula = Species ~ ., data = dt.train)
##
##
## C5.0 [Release 2.07 GPL Edition]      Wed Mar 11 21:12:14 2020
## -------------------------------
##
## Class specified by attribute `outcome'
##
```

```
## Read 105 cases (5 attributes) from undefined.data
##
## Decision tree:
##
## Petal.Length <= 1.9: setosa (35)
## Petal.Length > 1.9:
## :...Petal.Width > 1.7: virginica (33/1)
##     Petal.Width <= 1.7:
##     :...Petal.Length <= 4.9: versicolor (31/1)
##         Petal.Length > 4.9: virginica (6/2)
##
##
## Evaluation on training data (105 cases):
##
##      Decision Tree
##    ----------------
##    Size      Errors
##
##       4    4( 3.8%)   <<
##
##
##     (a)   (b)   (c)    <-classified as
##    ----  ----  ----
##      35                (a): class setosa
##            30     3    (b): class versicolor
##             1    36    (c): class virginica
##
##
##  Attribute usage:
##
##  100.00% Petal.Length
##   66.67% Petal.Width
##
##
## Time: 0.0 secs
```

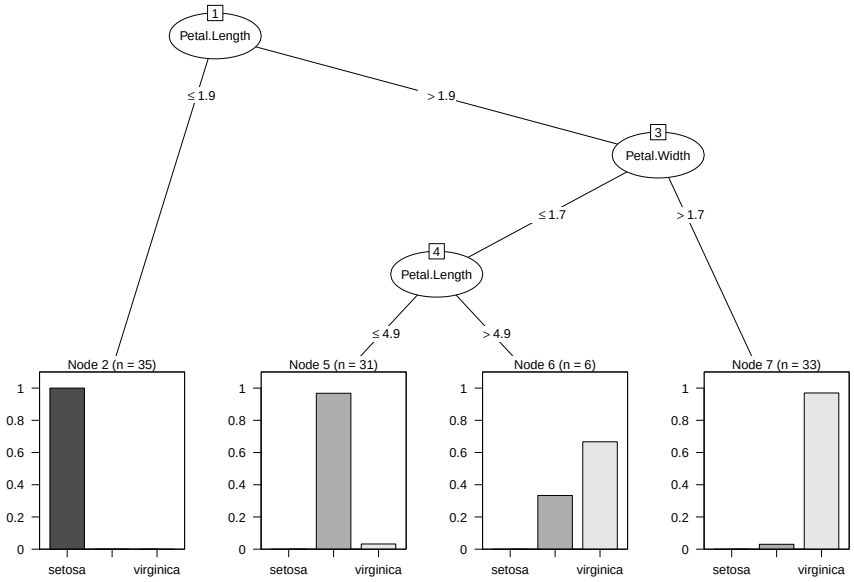
Otrzymujemy identyczne drzewo jak w przypadku zastosowania algorytmu `ctree`.

```
plot(tree4)
```

Dla pewności przeprowadzimy sprawdzenie na zbiorze testowym.

```
pred4 <- predict(tree4, newdata = dt.test)
tab <- table(predykcja = pred4, obserwacja = dt.test$Species)
tab
```

```
##             obserwacja
```



Rysunek 4.7: Wykres drzewa decyzyjnego zbudowanego metodą C5.0

```
## predykcja    setosa versicolor virginica
##   setosa         15          0         0
##   versicolor      0         17         0
##   virginica       0          0        13
```

Rozdział 5

Pochodne drzew decyzyjnych

Przykład zastosowania drzew decyzyjnych na zbiorze `iris` w poprzednich przykładach może skłaniać do przypuszczenia, że drzewa decyzyjne zawsze dobrze radzą sobie z predykcją wartości wynikowej. Niestety w przykładach nieco bardziej skomplikowanych, gdzie chociażby klasy zmiennej wynikowej nie są tak wyraźnie separowalne, drzewa decyzyjne wypadają gorzej w porównaniu z innymi modelami nadzorowanego uczenia maszynowego.

I tak u podstaw metod bazujących na prostych drzewach decyzyjnych stał pomysł, że skoro jedno drzewo nie ma wystarczających własności predykcyjnych, to może zastosowanie wielu drzew połączonych w pewien sposób poprawi je. Tak powstały metody *bagging*, *random forest* i *boosting*[1]. Należy zaznaczyć, że metody znajdują swoje zastosowanie również w innych modelach nadzorowanego uczenia maszynowego.

5.1 Bagging

Technika ta została wprowadzona przez Breiman (1996) i ma na celu zmniejszenie wariancji modelu pojedynczego drzewa. Podobnie jak technika *bootstrap*, w której statystyki są wyliczane na wielu próbach pobranych z tego samego rozkładu (próby), w metodzie bagging losuje się wiele prób ze zbioru uczącego (najczęściej poprzez wielokrotne losowanie próby o rozmiarze zbioru uczącego ze zwracaniem), a następnie dla każdej próby bootstrapowej buduje się drzewo. W ten sposób otrzymujemy B drzew decyzyjnych $\hat{f}^1(x), \hat{f}^2(x), \ldots, \hat{f}^B(x)$. Na koniec

[1] chyba tylko dla drugiej metody istniej dobre polskie tłumaczenie nazwy - las losowy

poprzez uśrednienie otrzymujemy model charakteryzujący się większą precyzją

$$\hat{f}_{bag}(x) = \frac{1}{B}\sum_{b=1}^{B}\hat{f}^b(x). \tag{5.1}$$

Ponieważ podczas budowy drzew na podstawie prób bootstrapowych nie kontrolujemy złożoności, to w rezultacie każde z drzew może charakteryzować się dużą wariancją. Poprzez uśrednianie wyników pojedynczych drzew otrzymujemy mniejsze obciążenie ale również przy dostatecznie dużej liczbie prób (B często liczy się w setkach, czy tysiącach) zmniejszamy wariancję "średniej" predykcji z drzew. Oczywiście metodę tą trzeba dostosować do zadań klasyfikacyjnych, ponieważ nie istnieje średnia klasyfikacji z wielu drzew. W miejsce średniej stosuje się modę, czyli wartość dominującą.

Przyjrzyjmy się jak maszyna losuje obserwacje ze zwracaniem

```
n <- NULL
m <- NULL
for(i in 1:1000){
    x <- sample(1:500, size = 500, replace = T)
    y <- setdiff(1:500, x)
    z <- unique(x)
    n[i] <- length(z)
    m[i] <- length(y)
}
mean(n)/500*100
```

```
## [1] 63.2802
```

```
mean(m)/500*100
```

```
## [1] 36.7198
```

Faktycznie uczenie modelu metodą bagging odbywa się średnio na 2/3 obserwacji zbioru uczącego wylosowanych do prób bootstrapowych, a pozostała 1/3 (ang. *out-of-bag*) jest wykorzystana do oceny jakości predykcji.

Niewątpliwą zaletą drzew decyzyjnych była ich łatwa interpretacja. W przypadku metody bagging jest ona znacznie utrudniona, ponieważ jej wynik składa się z agregacji wielu drzew. Można natomiast ocenić ważność predyktorów (ang. *variable importance*). I tak, przez obserwację spadku RSS dla baggingu regresyjnego przy zastosowaniu danego predyktora w podziałach drzewa i uśrednieniu wyniku otrzymamy wskaźnik ważności predyktora dużo lepszy niż dla pojedynczego drzewa. W przypadku baggingu klasyfikacyjnego w miejsce RSS stosujemy indeks Gini'ego.

Implementacja R-owa metody bagging znajduje się w pakiecie **ipred**, a funkcja do budowy modelu nazywa się `bagging` (Peters and Hothorn 2018). Można również stosować funkcję `randomForest` pakietu **randomForest** (Liaw and Wiener 2002) - powody takiego działania wyjaśnią się w podrozdziale Lasy losowe.

Przykład 5.1. Tym razem cel zadania jest regresyjny i polega na ustaleniu miary tendencji centralnej ceny mieszkań w Bostonie na podstawie zmiennych umieszczonych w zbiorze `Boston` pakietu **MASS** (Venables and Ripley 2002). Zmienną zależną będzie mediana cen mieszkań na przedmieściach Bostonu (`medv`).

```
library(MASS)
head(Boston)
```

```
##      crim zn indus chas   nox    rm  age    dis rad tax ptratio  black lstat
## 1 0.00632 18  2.31    0 0.538 6.575 65.2 4.0900   1 296    15.3 396.90  4.98
## 2 0.02731  0  7.07    0 0.469 6.421 78.9 4.9671   2 242    17.8 396.90  9.14
## 3 0.02729  0  7.07    0 0.469 7.185 61.1 4.9671   2 242    17.8 392.83  4.03
## 4 0.03237  0  2.18    0 0.458 6.998 45.8 6.0622   3 222    18.7 394.63  2.94
## 5 0.06905  0  2.18    0 0.458 7.147 54.2 6.0622   3 222    18.7 396.90  5.33
## 6 0.02985  0  2.18    0 0.458 6.430 58.7 6.0622   3 222    18.7 394.12  5.21
##   medv
## 1 24.0
## 2 21.6
## 3 34.7
## 4 33.4
## 5 36.2
## 6 28.7
```

```
set.seed(2019)
boston.train <- Boston %>%
    sample_frac(size = 2/3)
boston.test <- setdiff(Boston, boston.train)
```

Aby móc porównać wyniki predykcji z metody bagging, najpierw zostanie zbudowane jedno drzewo decyzyjne w oparciu o algorytm CART.

```
library(rpart)
library(rpart.plot)
boston.rpart <- rpart(medv~., data = boston.train)
x <- summary(boston.rpart)
```

```
## Call:
## rpart(formula = medv ~ ., data = boston.train)
##   n= 337
##
##           CP nsplit rel error    xerror       xstd
## 1 0.49839799      0 1.0000000 1.0086928 0.10259521
## 2 0.15725128      1 0.5016020 0.5442932 0.06125724
## 3 0.07485605      2 0.3443507 0.4031978 0.05139310
## 4 0.03672387      3 0.2694947 0.3127794 0.04599170
## 5 0.03552748      4 0.2327708 0.2974517 0.04560807
## 6 0.01695185      5 0.1972433 0.2553208 0.04022970
```

```
## 7 0.01422576      6 0.1802915 0.2713816 0.04099092
## 8 0.01103490      7 0.1660657 0.2744789 0.04107777
## 9 0.01000000      8 0.1550308 0.2720415 0.04119266
##
## Variable importance
##      rm   lstat   indus ptratio    crim     age     nox     dis      zn     tax
##      33      19       9       8       7       6       6       5       3       2
##     rad    chas
##       1       1
##
## Node number 1: 337 observations,    complexity param=0.498398
##   mean=22.69792, MSE=79.32964
##   left son=2 (286 obs) right son=3 (51 obs)
##   Primary splits:
##       rm      < 6.92     to the left,  improve=0.4983980, (0 missing)
##       lstat   < 9.725    to the right, improve=0.4424796, (0 missing)
##       indus   < 6.66     to the right, improve=0.2796065, (0 missing)
##       ptratio < 19.65    to the right, improve=0.2600149, (0 missing)
##       nox     < 0.6695   to the right, improve=0.2346383, (0 missing)
##   Surrogate splits:
##       ptratio < 14.55    to the right, agree=0.884, adj=0.235, (0 split)
##       lstat   < 4.915    to the right, agree=0.878, adj=0.196, (0 split)
##       zn      < 87.5     to the left,  agree=0.864, adj=0.098, (0 split)
##       indus   < 1.605    to the right, agree=0.864, adj=0.098, (0 split)
##       crim    < 0.013355 to the right, agree=0.852, adj=0.020, (0 split)
##
## Node number 2: 286 observations,    complexity param=0.1572513
##   mean=20.04266, MSE=37.17489
##   left son=4 (114 obs) right son=5 (172 obs)
##   Primary splits:
##       lstat   < 14.405   to the right, improve=0.3954065, (0 missing)
##       nox     < 0.6695   to the right, improve=0.3012249, (0 missing)
##       crim    < 8.37969  to the right, improve=0.2817286, (0 missing)
##       ptratio < 20.15    to the right, improve=0.2392532, (0 missing)
##       dis     < 2.4737   to the left,  improve=0.2295258, (0 missing)
##   Surrogate splits:
##       age   < 84.3     to the right, agree=0.808, adj=0.518, (0 split)
##       dis   < 2.23935  to the left,  agree=0.773, adj=0.430, (0 split)
##       crim  < 4.067905 to the right, agree=0.762, adj=0.404, (0 split)
##       nox   < 0.5765   to the right, agree=0.762, adj=0.404, (0 split)
##       indus < 16.57    to the right, agree=0.759, adj=0.395, (0 split)
##
## Node number 3: 51 observations,    complexity param=0.07485605
##   mean=37.58824, MSE=54.4677
##   left son=6 (34 obs) right son=7 (17 obs)
##   Primary splits:
```

```
##       rm      < 7.47     to the left,  improve=0.72041550, (0 missing)
##       lstat   < 3.99     to the right, improve=0.34223650, (0 missing)
##       ptratio < 15.05    to the right, improve=0.21227430, (0 missing)
##       rad     < 2.5      to the left,  improve=0.10053340, (0 missing)
##       tax     < 267      to the right, improve=0.07935891, (0 missing)
##   Surrogate splits:
##       lstat < 3.99     to the right, agree=0.824, adj=0.471, (0 split)
##       indus < 1.215    to the right, agree=0.706, adj=0.118, (0 split)
##       chas  < 0.5      to the left,  agree=0.706, adj=0.118, (0 split)
##       tax   < 225      to the right, agree=0.706, adj=0.118, (0 split)
##       crim  < 1.3713   to the left,  agree=0.686, adj=0.059, (0 split)
##
## Node number 4: 114 observations,    complexity param=0.03552748
##   mean=15.33333, MSE=21.50994
##   left son=8 (77 obs) right son=9 (37 obs)
##   Primary splits:
##       crim    < 0.69916  to the right, improve=0.3873341, (0 missing)
##       nox     < 0.6615   to the right, improve=0.3541892, (0 missing)
##       dis     < 2.3497   to the left,  improve=0.3182514, (0 missing)
##       ptratio < 19.45    to the right, improve=0.3102781, (0 missing)
##       tax     < 567.5    to the right, improve=0.2823826, (0 missing)
##   Surrogate splits:
##       ptratio < 19.95    to the right, agree=0.895, adj=0.676, (0 split)
##       indus   < 14.345   to the right, agree=0.868, adj=0.595, (0 split)
##       nox     < 0.5825   to the right, agree=0.868, adj=0.595, (0 split)
##       tax     < 397      to the right, agree=0.868, adj=0.595, (0 split)
##       rad     < 16       to the right, agree=0.860, adj=0.568, (0 split)
##
## Node number 5: 172 observations,    complexity param=0.03672387
##   mean=23.16395, MSE=23.11579
##   left son=10 (82 obs) right son=11 (90 obs)
##   Primary splits:
##       lstat   < 9.645    to the right, improve=0.24693150, (0 missing)
##       rm      < 6.543    to the left,  improve=0.17749260, (0 missing)
##       ptratio < 17.85    to the right, improve=0.07815189, (0 missing)
##       nox     < 0.5125   to the right, improve=0.07760816, (0 missing)
##       tax     < 267.5    to the right, improve=0.07238020, (0 missing)
##   Surrogate splits:
##       nox   < 0.5125   to the right, agree=0.756, adj=0.488, (0 split)
##       indus < 7.625    to the right, agree=0.750, adj=0.476, (0 split)
##       rm    < 6.26     to the left,  agree=0.738, adj=0.451, (0 split)
##       age   < 65.25    to the right, agree=0.727, adj=0.427, (0 split)
##       dis   < 3.8824   to the left,  agree=0.709, adj=0.390, (0 split)
##
## Node number 6: 34 observations
##   mean=33.15882, MSE=13.41419
```

```
##
## Node number 7: 17 observations
##   mean=46.44706, MSE=18.85661
##
## Node number 8: 77 observations,    complexity param=0.0110349
##   mean=13.33247, MSE=15.64998
##   left son=16 (37 obs) right son=17 (40 obs)
##   Primary splits:
##       lstat < 20.1     to the right, improve=0.24481010, (0 missing)
##       crim  < 15.718   to the right, improve=0.23250740, (0 missing)
##       dis   < 2.0037   to the left,  improve=0.17113480, (0 missing)
##       nox   < 0.6615   to the right, improve=0.11757680, (0 missing)
##       rm    < 5.5675   to the left,  improve=0.09054612, (0 missing)
##   Surrogate splits:
##       dis   < 1.9733   to the left,  agree=0.792, adj=0.568, (0 split)
##       rm    < 5.632    to the left,  agree=0.727, adj=0.432, (0 split)
##       age   < 95.35    to the right, agree=0.675, adj=0.324, (0 split)
##       crim  < 9.08499  to the right, agree=0.662, adj=0.297, (0 split)
##       black < 396.295  to the right, agree=0.623, adj=0.216, (0 split)
##
## Node number 9: 37 observations
##   mean=19.4973, MSE=8.034858
##
## Node number 10: 82 observations
##   mean=20.66098, MSE=6.55677
##
## Node number 11: 90 observations,    complexity param=0.01695185
##   mean=25.44444, MSE=27.29425
##   left son=22 (83 obs) right son=23 (7 obs)
##   Primary splits:
##       age   < 86.7     to the left,  improve=0.1844883, (0 missing)
##       lstat < 4.46     to the right, improve=0.1773076, (0 missing)
##       dis   < 3.0037   to the right, improve=0.1652768, (0 missing)
##       crim  < 0.628575 to the left,  improve=0.1203635, (0 missing)
##       nox   < 0.5585   to the left,  improve=0.1122403, (0 missing)
##   Surrogate splits:
##       nox     < 0.5585   to the left,  agree=0.978, adj=0.714, (0 split)
##       dis     < 2.1491   to the right, agree=0.978, adj=0.714, (0 split)
##       crim    < 0.643205 to the left,  agree=0.967, adj=0.571, (0 split)
##       indus   < 16.57    to the left,  agree=0.956, adj=0.429, (0 split)
##       ptratio < 14.75    to the right, agree=0.956, adj=0.429, (0 split)
##
## Node number 16: 37 observations
##   mean=11.2973, MSE=10.14026
##
## Node number 17: 40 observations
```

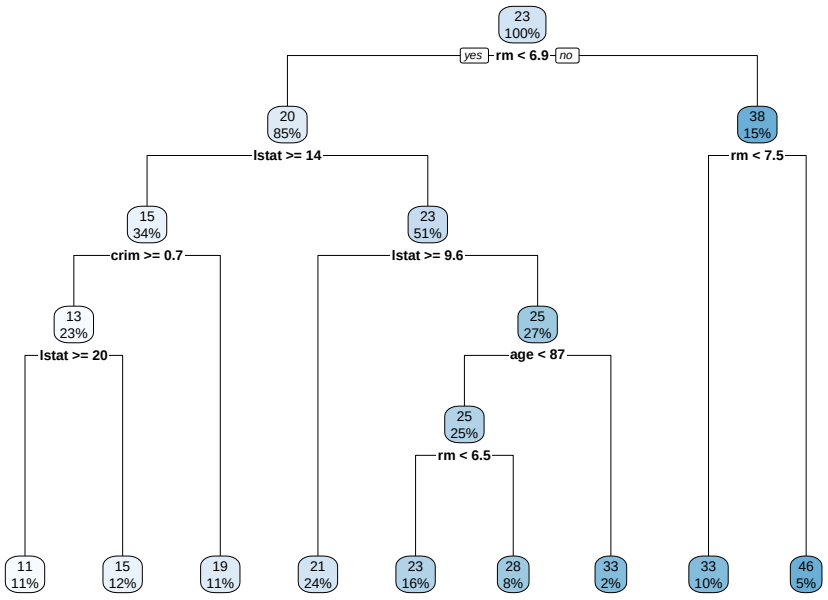
```
##   mean=15.215, MSE=13.37128
##
## Node number 22: 83 observations,    complexity param=0.01422576
##   mean=24.79277, MSE=13.56694
##   left son=44 (55 obs) right son=45 (28 obs)
##   Primary splits:
##       rm      < 6.543    to the left,  improve=0.3377388, (0 missing)
##       lstat   < 5.41     to the right, improve=0.2548210, (0 missing)
##       tax     < 267.5    to the right, improve=0.2210129, (0 missing)
##       ptratio < 18.05    to the right, improve=0.1394682, (0 missing)
##       dis     < 6.4889   to the right, improve=0.1125739, (0 missing)
##   Surrogate splits:
##       lstat   < 5.055    to the right, agree=0.783, adj=0.357, (0 split)
##       ptratio < 15.75    to the right, agree=0.723, adj=0.179, (0 split)
##       crim    < 0.39646  to the left,  agree=0.699, adj=0.107, (0 split)
##       chas    < 0.5      to the left,  agree=0.687, adj=0.071, (0 split)
##       age     < 74.15    to the left,  agree=0.687, adj=0.071, (0 split)
##
## Node number 23: 7 observations
##   mean=33.17143, MSE=125.3192
##
## Node number 44: 55 observations
##   mean=23.26545, MSE=8.880443
##
## Node number 45: 28 observations
##   mean=27.79286, MSE=9.189949
```

```
rpart.plot(boston.rpart)
```

Przycinamy drzewo...

```
printcp(boston.rpart)
```

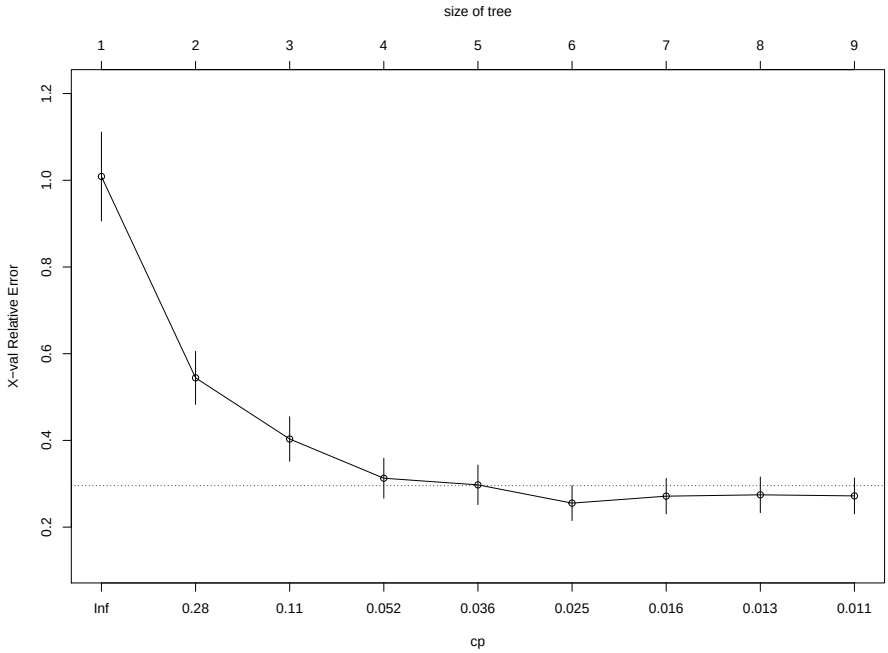
```
##
## Regression tree:
## rpart(formula = medv ~ ., data = boston.train)
##
## Variables actually used in tree construction:
## [1] age   crim  lstat rm
##
## Root node error: 26734/337 = 79.33
##
## n= 337
##
##         CP nsplit rel error  xerror     xstd
## 1 0.498398      0   1.00000 1.00869 0.102595
## 2 0.157251      1   0.50160 0.54429 0.061257
```



Rysunek 5.1: Drzewo regresyjne pełne

```
## 3 0.074856      2   0.34435 0.40320 0.051393
## 4 0.036724      3   0.26949 0.31278 0.045992
## 5 0.035527      4   0.23277 0.29745 0.045608
## 6 0.016952      5   0.19724 0.25532 0.040230
## 7 0.014226      6   0.18029 0.27138 0.040991
## 8 0.011035      7   0.16607 0.27448 0.041078
## 9 0.010000      8   0.15503 0.27204 0.041193
```

```
plotcp(boston.rpart)
```




```
boston.rpart2 <- prune(boston.rpart, cp = 0.016952)
```

```
rpart.plot(boston.rpart2)
```

Predykcja na podstawie drzewa na zbiorze testowym.

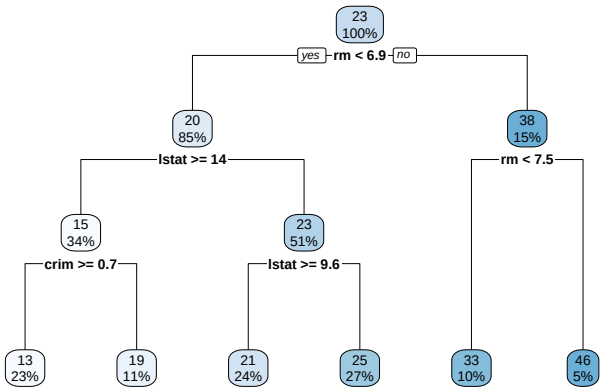
```
boston.pred <- predict(boston.rpart2, newdata = boston.test)
rmse <- function(pred, obs) sqrt(1/length(pred)*sum((pred-obs)^2))
rmse(boston.pred, boston.test$medv)
```

```
## [1] 5.830722
```

Teraz zbudujemy model metodą bagging.

```
library(randomForest)
boston.bag <- randomForest(medv~., data = boston.train,
                           mtry = ncol(boston.train)-1)
boston.bag
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = boston.train, mtry = ncol(boston.train) -      1)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
```



Rysunek 5.2: Drzewo regresyjne przycięte

```
##           Mean of squared residuals: 12.03374
##                     % Var explained: 84.83
```

Predykcja na podstawie modelu

```
boston.pred2 <- predict(boston.bag, newdata = boston.test)
rmse(boston.pred2, boston.test$medv)
```

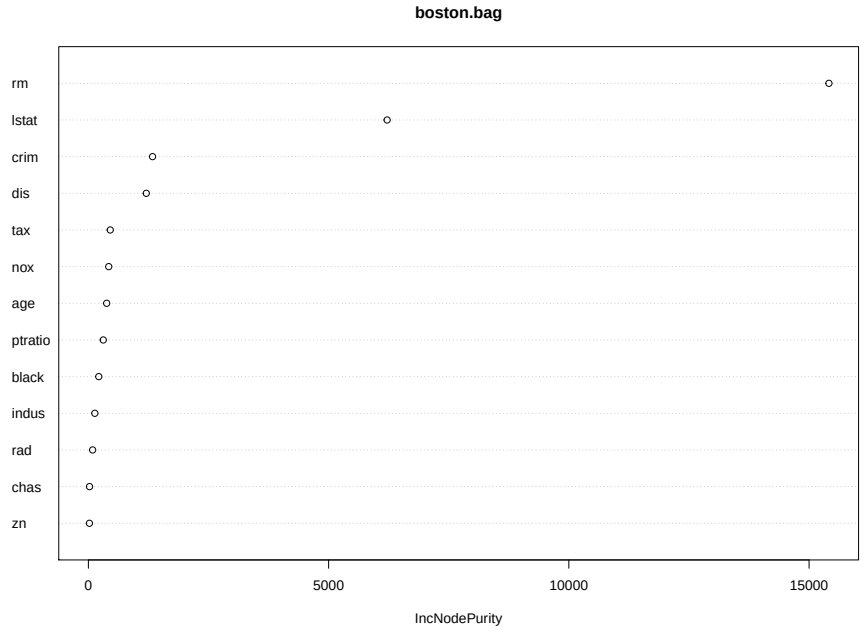
```
## [1] 4.359119
```

Zatem predykcja na podstawie modelu bagging jest nico lepsza niż z pojedynczego drzewa. Dodatkowo możemy ocenić ważność zmiennych użytych w budowie drzew.

```
varImpPlot(boston.bag)
```

```
importance(boston.bag)
```

```
##         IncNodePurity
## crim       1335.62584
## zn           21.35274
## indus       134.28748
## chas         24.07230
## nox         423.26229
```



Rysunek 5.3: Wykres ważności predyktorów

```
## rm        15413.69291
## age         380.78172
## dis        1204.86690
## rad          88.28151
## tax         454.99800
## ptratio     309.58412
## black       216.15512
## lstat      6217.95834
```

```
x$variable.importance
```

```
##          rm       lstat       indus     ptratio        crim         age
## 16276.30598  9170.91941  4427.10554  4039.00112  3412.53062  3170.82658
##         nox         dis          zn         tax         rad        chas
##  3063.70694  2681.24858  1306.29569   800.17910   539.07271   262.60146
##       black
##    63.78554
```

W porównaniu do ważności zmiennych dla pojedynczego drzewa widać pewne różnice.

5.2 Lasy losowe

Lasy losowe są uogólnieniem metody bagging, polegającą na losowaniu dla każdego drzewa wchodzącego w skład lasu m predyktorów spośród p dostępnych, a następnie budowaniu drzew z wykorzystaniem tylko tych predyktorów (Ho 1995). Dzięki temu za każdy razem drzewo jest budowane w oparciu o nowy zestaw cech (najczęściej przyjmujemy $m = \sqrt{p}$). W przypadku modeli bagging za każdym razem najsilniejszy predyktor wchodził w skład zbioru uczącego, a co za tym idzie również uczestniczył w tworzeniu reguł podziału. Wówczas wiele drzew zawierało reguły stosujące dany atrybut, a wtedy predykcje otrzymywane za pomocą drzew były skorelowane. Dlatego nawet duża liczba prób bootstrapowych nie zapewniała poprawy precyzji. Implementacja tej metody znajduje się w pakiecie **randomForest**.

Przykład 5.2. Kontynuując poprzedni przykład **??** możemy zbudować las losowy aby przekonać się czy nastąpi poprawa predykcji zmiennej wynikowej.

```
boston.rf <- randomForest(medv~., data = boston.train)
boston.rf
```

```
##
## Call:
##  randomForest(formula = medv ~ ., data = boston.train)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 12.05123
##                     % Var explained: 84.81
```

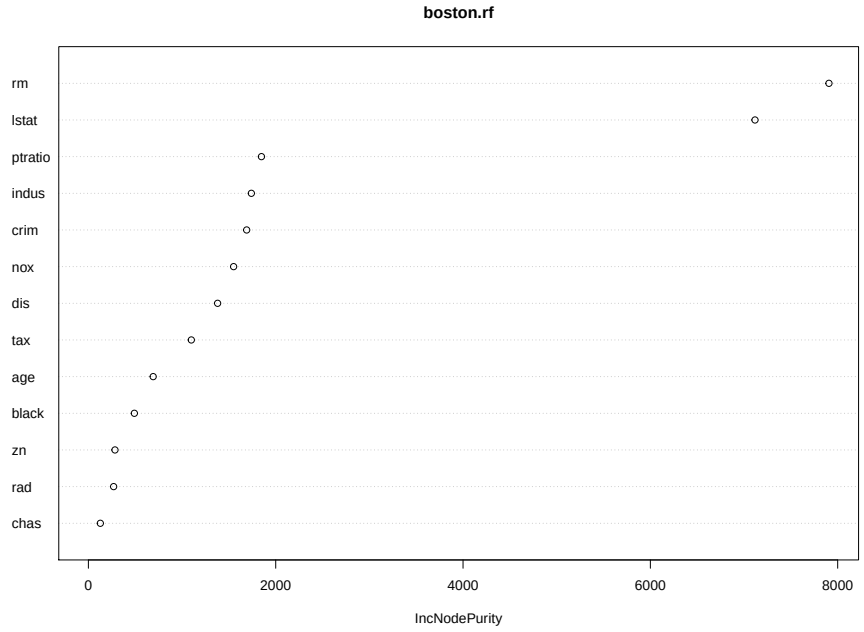
Porównanie MSE na próbach uczących pomiędzy lasem losowym i modelem bagging wypada nieco na korzyść bagging.

```
boston.pred3 <- predict(boston.rf, newdata = boston.test)
rmse(boston.pred3, boston.test$medv)
```

```
## [1] 3.79973
```

Ważność zmiennych również się nieco różni.

```
varImpPlot(boston.rf)
```



5.3 Boosting

Rozważania na temat metody *boosting* zaczęły się od pytań postawionych w publikacji Kearns and Valiant (1989), czy da się na podstawie na podstawie zbioru słabych modeli stworzyć jeden dobry? Odpowiedzi pozytywnej na nie udzielili, najpierw Schapire (1990), a potem Breiman (1998). W metodzie boosting nie stosuje się prób bootstrapowych ale odpowiednio modyfikuje się drzewo wyjściowe w kolejnych krokach na tym samym zbiorze uczącym. Algorytm dla drzewa regresyjnego jest następujący:

1. Ustal $\widehat{f}(x) = 0$ i $r_i = y_i$ dla każdego i w zbiorze uczącym.
2. Dla $b = 1, 2, \ldots, B$ powtarzaj:
   a) naucz drzewo $\widehat{f}^b$ o d regułach podziału (czyli $d+1$ liściach) na zbiorze (X_i, r_i),
   b) zaktualizuj drzewo do nowej "skurczonej" wersji
   $$\widehat{f}(x) \leftarrow \widehat{f}(x) + \lambda \widehat{f}^b(x), \tag{5.2}$$
   c) zaktualizuj reszty
   $$r_i \leftarrow r_i - \lambda \widehat{f}^b(x_i). \tag{5.3}$$
3. Wyznacz boosted model
$$\widehat{f}(x) = \sum_{b=1}^{B} \lambda \widehat{f}^b(x) \tag{5.4}$$

Uczenie drzew klasyfikacyjnego metoda boosting przebiega w podobny sposób. Wynik uczenia drzew metodą boosting zależy od trzech parametrów:

1. Liczby drzew B. W przeciwieństwie do metody bagging i lasów losowych, zbyt duże B może doprowadzić do przeuczenia modelu. B ustala się najczęściej na podstawie walidacji krzyżowej.
2. Parametru "kurczenia" (ang. *shrinkage*) λ. Kontroluje on szybkość uczenia się kolejnych drzew. Typowe wartości λ to 0.01 lub 0.001. Bardzo małe λ może wymagać dobrania większego B, aby zapewnić dobrą jakość predykcyjną modelu.
3. Liczby podziałów w drzewach d, która decyduje o złożoności drzewa. Bywa, że nawet $d = 1$ daje dobre rezultaty, ponieważ model wówczas uczy się powoli.

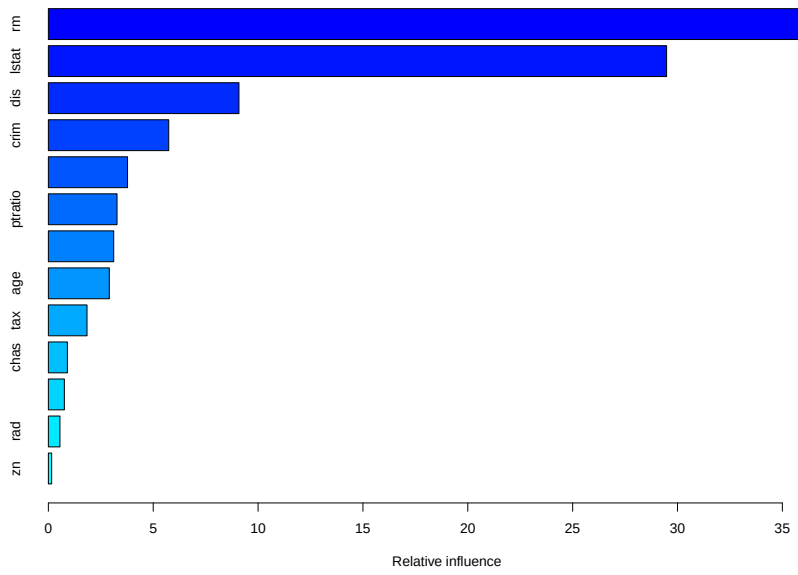
Implementację metody boosting można znaleźć w pakiecie **gbm** (Greenwell et al. 2019)

Przykład 5.3. Metodę boosting zastosujemy do zadania predykcji ceny mieszkań na przedmieściach Bostonu. Dobór parametrów modelu będzie arbitralny, więc niekoniecznie model będzie najlepiej dopasowany.

```
library(gbm)
boston.boost <- gbm(medv~., data = boston.train,
                    distribution = "gaussian",
                    n.trees = 5000,
                    interaction.depth = 2,
                    shrinkage = 0.01)
boston.boost
```

```
## gbm(formula = medv ~ ., distribution = "gaussian", data = boston.train,
##     n.trees = 5000, interaction.depth = 2, shrinkage = 0.01)
## A gradient boosted model with gaussian loss function.
## 5000 iterations were performed.
## There were 13 predictors of which 13 had non-zero influence.
```

```
summary(boston.boost)
```




```
##             var    rel.inf
## rm           rm 38.3955886
## lstat     lstat 29.4805422
## dis         dis  9.0886721
## crim       crim  5.7399540
## nox         nox  3.7754214
## ptratio ptratio  3.2740541
## black     black  3.1164954
## age         age  2.9063950
## tax         tax  1.8433918
## chas       chas  0.9067974
## indus     indus  0.7627923
## rad         rad  0.5523485
## zn           zn  0.1575472
```

Predykcja na podstawie metody boosting

```
boston.pred4 <- predict(boston.boost, newdata = boston.test, n.trees = 5000)
rmse(boston.pred4, boston.test$medv)
```

```
## [1] 3.801233
```

RMSE jest w tym przypadku nieco większe niż w lasach losowych ale sporo mniejsze niż w metodzie bagging. Wszystkie metody wzmacnianych drzew dają wyniki lepsze niż pojedyncze drzewa.

Rozdział 6

Klasyfikatory liniowe

Obszerną rodzinę klasyfikatorów stanowią modele liniowe (ang. *linear classification models*). Klasyfikacji w tej rodzinie technik dokonuje się na podstawie modeli funkcji kombinacji liniowej predyktorów. Jest to ujęcie parametryczne, w którym klasyfikacji nowej wartości dokonujemy na podstawie atrybutów obserwacji i wektora parametrów. Uczenie na podstawie zestawu treningowego polega na oszacowaniu parametrów modelu. W odróżnieniu od metod nieparametrycznych postać modelu tym razem jest znana. Każdy klasyfikator liniowy skład się z funkcji wewnętrznej (ang. *inner representation function*) i funkcji zewnętrznej (ang. *outer representation function*). Pierwsza jest funkcją rzeczywistą parametrów modelu i wartości atrybutów obserwacji

$$g(x) = F(\mathbf{a}(x), \mathbf{w}) = \sum_{i=0}^{p} w_i a_i(x) = \mathbf{w} \circ \mathbf{a}(x), \tag{6.1}$$

przyjmując, że $a_0(x) = 1$.

Funkcja zewnętrzna przyporządkowuje binarnie klasy na podstawie wartości funkcji wewnętrznej. Istnieją dwa główne typy tych klasyfikacji:

- brzegowa - przyjmujemy, że funkcje wewnętrzne tworzą granice zbiorów obserwacji różnych klas,
- probabilistyczna - bazująca na tym, że funkcje wewnętrzne mogą pośrednio wykazywać prawdopodobieństwo przynależności do danej klasy.

Pierwsza dzieli przestrzeń obserwacji za pomocą hiperpłaszczyzn na obszary jednorodne pod względem przynależności do klas. Druga jest próbą parametrycznej reprezentacji prawdopodobieństw przynależności do klas. Klasyfikacji na podstawie prawdopodobieństw można dokonać na różne sposoby, stosując:

- największe prawdopodobieństwo,
- funkcję najmniejszego kosztu błędnej klasyfikacji,

- krzywych ROC (ang. *Receiver Operating Characteristic* - o tym później).

Podejście brzegowe lub probabilistyczne prowadzi najczęściej do dwóch typów reprezentacji funkcji zewnętrznej:

- reprezentacji progowej (ang. *threshold representation*) - najczęściej przy podejściu brzegowym,
- reprezentacji logistycznej (ang. *logit representation*) - przy podejściu probabilistycznym.

6.1 Reprezentacja progowa

W przypadku klasyfikacji dwustanowej, dziedzina jest dzielona na dwa regiony (pozytywny i negatywny) poprzez porównanie funkcji zewnętrznej z wartością progową. Bez straty ogólności można sprawić, że będzie to wartość 0

$$h(x) = H(g(x)) = \begin{cases} 1, & \text{jeśli } g(x) \geq 0 \\ 0, & \text{w przeciwnym przypadku.} \end{cases} \tag{6.2}$$

Czasami używa się parametryzacji $\{-1, 1\}$. Przez porównanie $g(x)$ z 0 definiuje się hiperpłaszczyznę w p wymiarowej przestrzeni, która rozdziela dziedzinę na regiony pozytywne i negatywne. W tym ujęciu mówimy o liniowej separowalności obserwacji różnych klas, jeśli istnieje hiperpłaszczyzna je rozdzielająca.

6.2 Reprezentacja logitowa

Najbardziej popularną reprezentacją parametryczną stosowaną w klasyfikacji jest reprezentacja logitowa

$$¶(y = 1|x) = \frac{e^{g(x)}}{e^{g(x)} + 1}. \tag{6.3}$$

Wówczas $g(x)$ nie reprezentuje bezpośrednio $¶(y = 1|x)$ ale jego logit

$$g(x) = \text{logit}(¶(y = 1|x)), \tag{6.4}$$

gdzie $\text{logit}(p) = \ln \frac{p}{1-p}$. Dlatego właściwa postać reprezentacji jest następująca

$$¶(y = 1|x) = \text{logit}^{-1}(g(x)). \tag{6.5}$$

W ten sposób reprezentacja logitowa jest równoważna reprezentacji progowej, ponieważ

$$g(x) = \ln \frac{¶(y = 1|x)}{1 - ¶(y = 1|x)} = \ln \frac{¶(y = 1|x)}{¶(y = 0|x)} > 0. \tag{6.6}$$

Jednak zaletą reprezentacji logitowej, w porównaniu do progowej, jest to, że można wyznaczyć prawdopodobieństwa przynależności do obu klas. W przypadku klasyfikacji wielostanowej uczymy tyle funkcji h ile jest klas.

6.3 Wady klasyfikatorów liniowych

- tylko w przypadku prostych funkcji wewnętrznych jesteśmy w stanie ocenić wpływ poszczególnych predykorów na klasyfikację,
- jakość predykcji zależy od doboru funkcji wewnętrznej (liniowa w ścisłym sensie jest najczęściej niewystarczająca),
- nie jest w stanie klasyfikować poprawnie stanów (nie jest liniowo separowalna) w zagadnieniach typu XOR.

Rozdział 7

Regresja logistyczna

7.1 Model

Regresja logistyczna (ang. *logistic regression*) jest techniką z rodziny klasyfikatorów liniowych z reprezentacją logistyczną, a formalnie należy do rodziny uogólnionych modeli liniowych (GLM). Stosowana jest wówczas, gdy zmienna wynikowa posiada dwa stany (sukces i porażka), kodowane najczęściej za pomocą 1 i 0. W tej metodzie modelowane jest warunkowe prawdopodobieństwo sukcesu za pomocą kombinacji liniowej predyktorów X.

Ogólna postać modelu

$$Y \sim B(1,p) \tag{7.1}$$

$$p(X) = \mathrm{E}(Y|X) = \frac{\exp(\beta X)}{1+\exp(\beta X)}, \tag{7.2}$$

gdzie $B(1,p)$ jest rozkładem dwumianowym o prawdopodobieństwie sukcesu p, a βX oznacza kombinację liniową parametrów modelu i wartości zmiennych niezależnych, przyjmując, że $x_0 = 1$. Jako funkcji łączącej (czyli opisującej związek między kombinacją liniową predyktorów i prawdopodobieństwem sukcesu) użyto *logitu*. Pozwala on na wygodną interpretację wyników w terminach szans.

Szansą (ang. *odds*) nazywamy stosunek prawdopodobieństwa sukcesu do prawdopodobieństwa porażki

$$o = \frac{p}{1-p}. \tag{7.3}$$

Ponieważ będziemy przyjmowali, że $p \in (0,1)$, to $o \in (0,\infty)$, a jej logarytm należy do przedziału $(-\infty,\infty)$.

Zatem logarytm szansy jest kombinacją liniową predyktorów

$$\log\left[\frac{p(X)}{1-p(X)}\right] = \beta_0 + \beta_1 x_1 + ... + \beta_d x_d. \tag{7.4}$$

7.2 Estymacja parametrów modelu

Estymacji parametrów modelu logistycznego dokonujemy za pomocą metody największej wiarogodności. Funkcja wiarogodności w tym przypadku przyjmuje postać

$$L(X_1, \dots, X_n, \beta) = \prod_{i=1}^{n} p(X_i)_i^Y [1 - p(X_i)]^{1-Y_i}, \tag{7.5}$$

gdzie wektor β jest uwikłany w funkcji $p(X_i)$. Maksymalizacji dokonujemy raczej po nałożeniu na funkcję wiarogodności logarytmu, bo to ułatwia szukanie ekstremum.

$$\log L(X_1, \dots, X_n, \beta) = \sum_{i=1}^{n} (Y_i \log p(X_i) + (1 - Y_i) \log(1 - p(X_i))). \tag{7.6}$$

7.3 Interpretacja

Interpretacja (lat. *ceteris paribus* - "inne takie samo") poszczególnych parametrów modelu jest następująca:

- jeśli $b_i > 0$ - to zmienna x_i ma wpływ stymulujący pojawienie się sukcesu,
- jeśli $b_i < 0$ - to zmienna x_i ma wpływ ograniczający pojawienie się sukcesu,
- jeśli $b_i = 0$ - to zmienna x_i nie ma wpływu na pojawienie się sukcesu.

Iloraz szans (ang. *odds ratio*) stosuje się w przypadku porównywania dwóch klas obserwacji. Jest on jak sama nazwa wskazuje ilorazem szans zajścia sukcesu w obu klasach

$$OR = \frac{p_1}{1 - p_1} \frac{1 - p_2}{p_2}, \tag{7.7}$$

gdzie p_i oznacza zajście sukcesu w i-tej klasie.

Interpretujemy go następująco:

- jeśli $OR > 1$ - to w pierwszej grupie zajście sukcesu jest bardziej prawdopodobne,
- jeśli $OR < 1$ - to w drugiej grupie zajście sukcesu jest bardziej prawdopodobne,
- jeśli $OR = 1$ - to w obu grupach zajście sukcesu jest jednakowo prawdopodobne.

Przykład 7.1. Jako ilustrację działania regresji logistycznej użyjemy modelu dla danych ze zbioru `Default` pakietu `ISLR`.

```
library(ISLR)
head(Default)
```

```
##   default student   balance    income
## 1      No      No  729.5265 44361.625
## 2      No     Yes  817.1804 12106.135
```

```
## 3      No      No 1073.5492 31767.139
## 4      No      No  529.2506 35704.494
## 5      No      No  785.6559 38463.496
## 6      No     Yes  919.5885  7491.559
```

Zmienną zależną jest `default`, a pozostałe są predyktorami. najpierw dokonamy podziału próby na ucząca i testową, a następnie zbudujemy model.

```
set.seed(2019)
ind <- sample(1:nrow(Default), size = 2/3*nrow(Default))
dt.ucz <- Default[ind,]
dt.test <- Default[-ind,]
mod.logit <- glm(default~., dt.ucz, family = binomial("logit"))
summary(mod.logit)
```

```
##
## Call:
## glm(formula = default ~ ., family = binomial("logit"), data = dt.ucz)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.1913  -0.1410  -0.0537  -0.0192   3.7527
##
## Coefficients:
##               Estimate Std. Error z value Pr(>|z|)
## (Intercept) -1.128e+01  6.169e-01 -18.287   <2e-16 ***
## studentYes  -4.627e-01  2.862e-01  -1.617    0.106
## balance      5.830e-03  2.883e-04  20.221   <2e-16 ***
## income       9.460e-06  9.833e-06   0.962    0.336
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 1967.2  on 6665  degrees of freedom
## Residual deviance: 1046.5  on 6662  degrees of freedom
## AIC: 1054.5
##
## Number of Fisher Scoring iterations: 8
```

Tylko `income` nie ma żadnego wpływu na prawdopodobieństwo stanu `Yes` zmiennej `default`. Zmienna `balance` wpływa stymulująco na prawdopodobieństwo pojawienia się sukcesu. Natomiast jeśli badana osoba jest studentem (`studentYes`), to ma wpływ ograniczający na pojawienie się sukcesu. Chcąc porównać dwie grupy obserwacji, przykładowo studentów z nie-studentami, możemy wykorzystać iloraz szans.

```
exp(cbind(OR = coef(mod.logit), confint(mod.logit))) %>%
    kable(digits = 4)
```

	OR	2.5 %	97.5 %
(Intercept)	0.0000	0.0000	0.0000
studentYes	0.6296	0.3598	1.1060
balance	1.0058	1.0053	1.0064
income	1.0000	1.0000	1.0000

Z powyższej tabeli wynika, że bycie studentem zmniejsza szanse na `Yes` w zmiennej `default` o około 37% (w stosunku do nie-studentów). Natomiast wzrost zmiennej `balance` przy zachowaniu pozostałych zmiennych na tym samym poziomie skutkuje wzrostem szans na `Yes` o około 0.6%.

Chcąc przeprowadzić predykcję na podstawie modelu dla ustalonych wartości cech (np. `student = Yes`, `balance = $1000` i `income = $40000`) postępujemy następująco

```
dt.new <- data.frame(student = "Yes", balance = 1000, income = 40000)
predict(mod.logit, newdata = dt.new, type = "response")
```

```
##           1
## 0.003931398
```

Otrzymany wynik jest oszacowanym prawdopodobieństwem warunkowym wystąpienia sukcesu (`default = Yes`). Widać zatem, że poziomy badanych cech sprzyjają raczej porażce.

Jeśli chcemy sprawdzić jakość klasyfikacji na zbiorze testowym, to musimy ustalić na jakim poziomie prawdopodobieństwa będziemy uznawać obserwację za sukces. W zależności od tego, na predykcji jakiego stanu zależy nam bardziej, możemy różnie dobierać ten próg (bez żadnych dodatkowych przesłanek najczęściej jest to 0.5).

```
pred <- predict(mod.logit, newdata = dt.test, type = "response")
pred.class <- ifelse(pred > 0.5, "Yes", "No")
(tab <- table(pred.class, dt.test$default))
```

```
##
## pred.class   No  Yes
##        No  3211   71
##        Yes   15   37
```

```
(acc <- sum(diag(prop.table(tab))))
```

```
## [1] 0.9742052
```

Klasyfikacja na poziomie 97% wskazuje na dobre dopasowanie modelu.