**Sztuczna Inteligencja 2023/24**

**Uczenie maszynowe – drzewa decyzyjne (laboratorium).**

# Karolina Dbaj, 09.07.2024

Do uczenia maszynowego metodą drzew decyzyjnych wybrano bazę **Student Performance Data Set**, która zawiera dane dotyczące **wyników w nauce studentów**. Celem zadania jest klasyfikacja obserwacji względem kryterium **czy student zdał, czy nie zdał egzaminu końcowego**. Zbiór danych zawiera **1044** próbek (obserwacji). Każda próbka zawiera **31** cech (tj. zmiennych przewidujących). Zmienna wyjaśniana (zależna przewidywana) o nazwie **G3** posiada **dwie** różne wartości: **fail, pass** stanowiące kryterium klasyfikacji. Wartości tej zmiennej mają następującą interpretację: **fail – student nie zdał, pass – student zdał egzamin końcowy**.

Nazwy i interpretację (krótki opis) zmiennych przewidujących (tj. cech) podaję poniżej w tabeli:

|  |  |
| --- | --- |
| school | szkoła |
| sex | Płeć |
| age | Wiek |
| address | adres |
| fansuze | wielkość rodziny |
| Pstatus | status rodziców |
| Medu | wykształcenie matki |
| Fedu | wykształcenie ojca |
| Mjob | zawód matki |
| Fjob | zawód ojca |
| reason | powód wyboru szkoły |
| guardian | opiekun |
| traveltime | czas dojazdu do szkoły |
| studytime | czas nauki w tygodniu |
| failures | liczba niezdanych przedmiotów |
| schoolsup | dodatkowe wsparcie edukacyjne |
| famsup | wsparcie rodziny |
| paid | dodatkowe płatne zajęcia |
| activities | zajęcia dodatkowe |
| nursery | uczęszczanie do przedszkola |
| higher | chęć podjęcia studiów |
| internet | dostęp do internetu |
| romantic | status romantyczny |
| famrel | relacje w rodzinie |
| freetime | czas wolny po szkole |
| goout | wyjścia z przyjaciółmi |
| Dalc | spożycie alkoholu w dni powszednie |
| Walc | spożycie alkoholu w weekend |
| health | stan zdrowia |
| absences | liczba nieobecności |

Do uczenia maszynowego wybrano algorytm **rpart** drzew decyzyjnych zaimplementowany w języku R na podstawie algorytmu **CART** (Classification And Regression Trees). Algorytm może być użyty zarówno do klasyfikacji jak i regresji – my wykorzystamy tylko klasyfikację.

Poniżej podano opis kolejnych kroków procesu tworzenia modelu uczenia maszynowego wraz z kodem w R oraz jego weryfikacji.

1. Wczytujemy zbiór danych do postaci Data Frame

1. Sys.setenv(LANG = "en")

2. library(dplyr)

3. library(tidyr)

4. library(rpart)

5. library(rpart.plot)

6. library(caret)

7.

8. d1 <- read.table("C:/Users/petit/Desktop/repos/uo/rok 3/Sztuczna inteligencja/DrzewaDecyzyjne - students/student-mat.csv", sep = ";", header = TRUE)

9. d2 <- read.table("C:/Users/petit/Desktop/repos/uo/rok 3/Sztuczna inteligencja/DrzewaDecyzyjne - students/student-por.csv", sep = ";", header = TRUE)

10.

11. d1\_d2 <- bind\_rows(d1, d2) #merge two dataframes

12. d1\_d2 <- select(d1\_d2, -G1, -G2) #remove G1 and G2

13. d1\_d2$G3 <- factor(ifelse(d1\_d2$G3 >= 10, 1, 0), labels = c("fail", "pass")) #replace marks on scale 1..20 to binary pass >=10 or fail <10

14. str(d1\_d2) #check for data type

15. d1\_d2 <- mutate\_if(d1\_d2, is.character, as.factor) #replace character type of data with factor type of data

**2.** Sprawdzamy czy dane są kompletne oraz jakiego są typu. Algorytm rpart wykorzystuje tzw. „surrogate variables” do zastępowania brakujących danych. Jeśli brakuje jakiejś danej dla użytej cechy, to dla tej próbki zostaje wykorzystana kolejna w rankingu cecha, gdzie ta dana jest obecna. Dlatego w przypadku korzystania z algorytmu rpart nie ma potrzeby uzupełniania brakujących danych.

Algorytm rpart wykorzystuje zmienne typu numerycznego lub factors. Zmienne innych typów należy zmienić na jeden z obsługiwanych typów. Data Frame zawiera zmienne następujących typów:

> str(d1\_d2) #check for data type

'data.frame': 1044 obs. of 31 variables:

$ school : chr "GP" "GP" "GP" "GP" ...

$ sex : chr "F" "F" "F" "F" ...

$ age : int 18 17 15 15 16 16 16 17 15 15 ...

$ address : chr "U" "U" "U" "U" ...

$ famsize : chr "GT3" "GT3" "LE3" "GT3" ...

$ Pstatus : chr "A" "T" "T" "T" ...

$ Medu : int 4 1 1 4 3 4 2 4 3 3 ...

$ Fedu : int 4 1 1 2 3 3 2 4 2 4 ...

$ Mjob : chr "at\_home" "at\_home" "at\_home" "health" ...

$ Fjob : chr "teacher" "other" "other" "services" ...

$ reason : chr "course" "course" "other" "home" ...

$ guardian : chr "mother" "father" "mother" "mother" ...

$ traveltime: int 2 1 1 1 1 1 1 2 1 1 ...

$ studytime : int 2 2 2 3 2 2 2 2 2 2 ...

$ failures : int 0 0 3 0 0 0 0 0 0 0 ...

$ schoolsup : chr "yes" "no" "yes" "no" ...

$ famsup : chr "no" "yes" "no" "yes" ...

$ paid : chr "no" "no" "yes" "yes" ...

$ activities: chr "no" "no" "no" "yes" ...

$ nursery : chr "yes" "no" "yes" "yes" ...

$ higher : chr "yes" "yes" "yes" "yes" ...

$ internet : chr "no" "yes" "yes" "yes" ...

$ romantic : chr "no" "no" "no" "yes" ...

$ famrel : int 4 5 4 3 4 5 4 4 4 5 ...

$ freetime : int 3 3 3 2 3 4 4 1 2 5 ...

$ goout : int 4 3 2 2 2 2 4 4 2 1 ...

$ Dalc : int 1 1 2 1 1 1 1 1 1 1 ...

$ Walc : int 1 1 3 1 2 2 1 1 1 1 ...

$ health : int 3 3 3 5 5 5 3 1 1 5 ...

$ absences : int 6 4 10 2 4 10 0 6 0 0 ...

$ G3 : Factor w/ 2 levels "fail","pass": 1 1 2 2 2 2 2 1 2 2 ...

W związku z tym musimy zmienić typ zmiennych: school, sex, address, famsize, Pstatus, Mjob, Fjob, reason, guardian, schoolsup, famsup, paid, activities, nursery, higher, internet, romantic na obsługiwany typ za pomocą kodu:

1. d1\_d2 <- mutate\_if(d1\_d2, is.character, as.factor)

2. str(d1\_d2)

Po przeprowadzonych operacjach na danych otrzymujemy Data Frame w następującej postaci:

> str(d1\_d2)

'data.frame': 1044 obs. of 31 variables:

$ school : Factor w/ 2 levels "GP","MS": 1 1 1 1 1 1 1 1 1 1 ...

$ sex : Factor w/ 2 levels "F","M": 1 1 1 1 1 2 2 1 2 2 ...

$ age : int 18 17 15 15 16 16 16 17 15 15 ...

$ address : Factor w/ 2 levels "R","U": 2 2 2 2 2 2 2 2 2 2 ...

$ famsize : Factor w/ 2 levels "GT3","LE3": 1 1 2 1 1 2 2 1 2 1 ...

$ Pstatus : Factor w/ 2 levels "A","T": 1 2 2 2 2 2 2 1 1 2 ...

$ Medu : int 4 1 1 4 3 4 2 4 3 3 ...

$ Fedu : int 4 1 1 2 3 3 2 4 2 4 ...

$ Mjob : Factor w/ 5 levels "at\_home","health",..: 1 1 1 2 3 4 3 3 4 3 ...

$ Fjob : Factor w/ 5 levels "at\_home","health",..: 5 3 3 4 3 3 3 5 3 3 ...

$ reason : Factor w/ 4 levels "course","home",..: 1 1 3 2 2 4 2 2 2 2 ...

$ guardian : Factor w/ 3 levels "father","mother",..: 2 1 2 2 1 2 2 2 2 2 ...

$ traveltime: int 2 1 1 1 1 1 1 2 1 1 ...

$ studytime : int 2 2 2 3 2 2 2 2 2 2 ...

$ failures : int 0 0 3 0 0 0 0 0 0 0 ...

$ schoolsup : Factor w/ 2 levels "no","yes": 2 1 2 1 1 1 1 2 1 1 ...

$ famsup : Factor w/ 2 levels "no","yes": 1 2 1 2 2 2 1 2 2 2 ...

$ paid : Factor w/ 2 levels "no","yes": 1 1 2 2 2 2 1 1 2 2 ...

$ activities: Factor w/ 2 levels "no","yes": 1 1 1 2 1 2 1 1 1 2 ...

$ nursery : Factor w/ 2 levels "no","yes": 2 1 2 2 2 2 2 2 2 2 ...

$ higher : Factor w/ 2 levels "no","yes": 2 2 2 2 2 2 2 2 2 2 ...

$ internet : Factor w/ 2 levels "no","yes": 1 2 2 2 1 2 2 1 2 2 ...

$ romantic : Factor w/ 2 levels "no","yes": 1 1 1 2 1 1 1 1 1 1 ...

$ famrel : int 4 5 4 3 4 5 4 4 4 5 ...

$ freetime : int 3 3 3 2 3 4 4 1 2 5 ...

$ goout : int 4 3 2 2 2 2 4 4 2 1 ...

$ Dalc : int 1 1 2 1 1 1 1 1 1 1 ...

$ Walc : int 1 1 3 1 2 2 1 1 1 1 ...

$ health : int 3 3 3 5 5 5 3 1 1 5 ...

$ absences : int 6 4 10 2 4 10 0 6 0 0 ...

$ G3 : Factor w/ 2 levels "fail","pass": 1 1 2 2 2 2 2 1 2 2 ...

**3.** Nasze dane zawierają następującą liczbę próbek każdej z klas:

fail: 229 próbek

pass: 815 próbek

W związku z tym nasze dane NIE SĄ zbalansowane. Dzielimy nasz zbiór danych na dwa podzbiory: zbiór treningowy: 75% danych i zbiór testowy: 25% danych. Zbiór treningowy będzie nam służył do konstrukcji modelu ML, a testowy do jego oceny. Poniżej przedstawiono kod do podziału danych na dwa podzbiory.

1. set.seed(3456)

2. trainIndex <- createDataPartition(d1\_d2$G3, p = 0.75, list = FALSE, times = 1)

3. sTrain <- d1\_d2[trainIndex,]

4. sTest <- d1\_d2[-trainIndex,]

Zweryfikowaliśmy, że po podziale (zastosowaliśmy stratyfikację) procentowa zawartość próbek różnych klas jest podobna jak w oryginalnym zbiorze.

**4.** Następnie wykorzystując kod przedstawiony poniżej:

1. dt\_control <- rpart.control(maxdepth = 25, xval = 10, cp = 0)

2. d1\_tree <- rpart(G3~., data = sTrain, method = "class", control = dt\_control, minsplit = 20)

dokonaliśmy konstrukcji drzewa decyzyjnego zgodnie z algorytmem rpart (CART) wykorzystując zbiór treningowy. Przy walidacji wykorzystano technikę sprawdzianu krzyżowego k-krotnego, przy czym k ustaliliśmy na 10 za pomocą zmiennej xval .

**5.** Po utworzeniu drzewa należy go przyciąć (prune) aby uniknąć nadmiernego dopasowania (overfitting). Robimy to korzystając z parametru cp oraz błędu względnego sprawdzianu krzyżowego xerror. Parametr cp zwany „complexity parameter” mówi nam jaka wymagana jest minimalna poprawa w modelu przy dodawaniu węzłów. Minimalną poprawę szacuje się na podstawie źle sklasyfikowanych próbek w liściach oraz liczby węzłów. Jeśli przy dodawaniu nowych liści liczba złych klasyfikacji nie będzie odpowiednio szybko malała (co jest szacowane wartością parametru cp) to drzewo nie będzie dalej konstruowane. Możemy też określić maksymalny rozmiar drzewa za pomocą parametru maxdepth. Poniżej przedstawiono tabelkę podającą wartości cp oraz odpowiadające im błędy walidacji krzyżowej (xerror) wraz z rozmiarem drzewa (tj. liczba węzłów, nsplit).

> printcp(d1\_tree)

Classification tree:

rpart(formula = G3 ~ ., data = sTrain, method = "class", control = dt\_control,

minsplit = 20)

Variables actually used in tree construction:

[1] absences address failures famrel Fedu Fjob goout higher Medu Mjob paid school sex

Root node error: 173/784 = 0.22066

n= 784

CP nsplit rel error xerror xstd

1 0.1271676 0 1.00000 1.00000 0.067118

2 0.0462428 1 0.87283 0.94220 0.065680

3 0.0144509 3 0.78035 0.87861 0.063985

4 0.0115607 5 0.75145 0.93064 0.065381

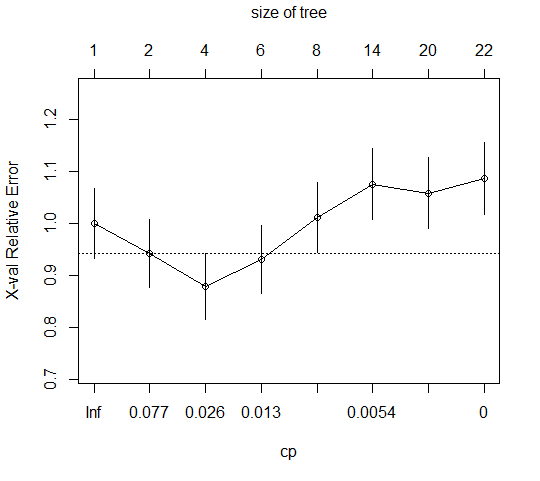
5 0.0077071 7 0.72832 1.01156 0.067394

6 0.0038536 13 0.68208 1.07514 0.068850

7 0.0028902 19 0.65896 1.05780 0.068463

8 0.0000000 21 0.65318 1.08671 0.069103

Na podstawie otrzymanych danych poniżej przedstawiono wykres błędu xerror w funkcji parametru złożoności cp.



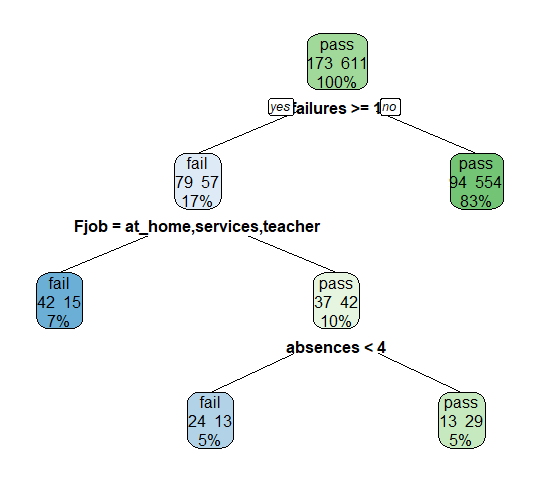
Na wykresie zaznaczono poziomą linią przerywaną wartość najmniejszego błędu powiększonego o odchylenie standardowe xstd. Punkty poniżej tej linii są dobrymi kandydatami na końcowe drzewo po przycięciu. Drzewa przycinamy za pomocą kodu:

1. d1\_tree <- prune(d1\_tree, cp = 0.02)

Po przycięciu drzewo możemy go narysować za pomocą kodu:

1. rpart.plot(d1\_tree, extra=101, fallen.leaves = FALSE, tweak = 1.0, varlen = 10, faclen = 10)

Poniżej przedstawiono przycięte drzewo dla wybranych wartości cp = 0.02.

****

**6**. Na zakończenie należy przetestować nasze drzewo wykorzystując zbiór testowy. Robimy to za pomocą kodu:

1. d1\_tree\_predict <- predict(d1\_tree, sTest, type="class")

2. confusionMatrix(table(d1\_tree\_predict, sTest$G3))

Poniżej przedstawiono wyniki testów dla wcześniej narysowanego drzewa.

> confusionMatrix(table(d1\_tree\_predict, sTest$G3))

Confusion Matrix and Statistics

d1\_tree\_predict fail pass

fail 13 18

pass 44 185

Accuracy : 0.7615

95% CI : (0.705, 0.812)

No Information Rate : 0.7808

P-Value [Acc > NIR] : 0.796446

Kappa : 0.1668

Mcnemar's Test P-Value : 0.001498

Sensitivity : 0.2281

Specificity : 0.9113

Pos Pred Value : 0.4194

Neg Pred Value : 0.8079

Prevalence : 0.2192

Detection Rate : 0.0500

Detection Prevalence : 0.1192

Balanced Accuracy : 0.5697

'Positive' Class : fail