**Sztuczna Inteligencja 2023/24**

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

**Błażej Domagała, 07.07.2024**

Do uczenia maszynowego metodą drzew decyzyjnych wybrano bazę Mushroom, która zawiera dane dotyczące klasyfikacji grzybów jako jadalne lub trujące. Celem zadania jest klasyfikacja obserwacji względem kryterium czy grzyb jest jadalny czy trujący. Zbiór danych zawiera 8124 próbek (obserwacji). Każda próbka zawiera 7 cech (tj. zmiennych przewidujących). Zmienna wyjaśniana (zależna, przewidywana) o nazwie class posiada dwie różne wartości: edible, poisonous stanowiące kryterium klasyfikacji. Wartości tej zmiennej mają następującą interpretację: edible - jadalny, poisonous - trujący.

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

|  |  |  |  |
| --- | --- | --- | --- |
| |  | | --- | | cap.shape | | |  | | --- | | Kształt kapelusza | |
| |  | | --- | | cap.surface | | |  | | --- | | Powierzchnia kapelusza | |
| |  |  | | --- | --- | | cap.color |  | | |  | | --- | | Kolor kapelusza | |
| |  |  | | --- | --- | | odor |  | | |  | | --- | | Zapach | |
| |  |  | | --- | --- | | population |  | | Populacja |
| |  |  | | --- | --- | | habitat |  | | Siedlisko |

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. grzyby <- read.csv("mushroom.csv")

**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(grzyby)

'data.frame': 8124 obs. of 7 variables:

$ class : chr "poisonous" "edible" "edible" "poisonous" ...

$ cap.shape : chr "convex" "convex" "bell" "convex" ...

$ cap.surface: chr "smooth" "smooth" "smooth" "scaly" ...

$ cap.color : chr "brown" "yellow" "white" "white" ...

$ odor : chr "pungent" "almond" "anise" "pungent" ...

$ population : chr "scattered" "numerous" "numerous" "scattered" ...

$ habitat : chr "urban" "grasses" "meadows" "urban" ...

W związku z tym musimy zmienić typ zmiennych: class, cap.shape, cap.surface, cap.color, odor, population, habitat na obsługiwany typ za pomocą kodu:

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

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

> str(grzyby)

'data.frame': 8124 obs. of 7 variables:

$ class : Factor w/ 2 levels "edible","poisonous": 2 1 1 2 1 1 1 1 2 1 ...

$ cap.shape : Factor w/ 6 levels "bell","conical",..: 3 3 1 3 3 3 1 1 3 1 ...

$ cap.surface: Factor w/ 4 levels "fibrous","grooves",..: 4 4 4 3 4 3 4 3 3 4 ...

$ cap.color : Factor w/ 10 levels "brown","buff",..: 1 10 9 9 4 10 9 9 9 10 ...

$ odor : Factor w/ 9 levels "almond","anise",..: 8 1 2 8 7 1 1 2 8 1 ...

$ population : Factor w/ 6 levels "abundant","clustered",..: 4 3 3 4 1 3 3 4 5 4 ...

$ habitat : Factor w/ 7 levels "grasses","leaves",..: 5 1 3 5 1 1 3 3 1 3 ...

**3.** Nasze dane zawierają następującą liczbę próbek każdej z klas: 4208 jadalnych, 3916 trujących, w związku z tym nasze dane 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. # Ustawienie ziarna losowego dla powtarzalności wyników

2. set.seed(1456)

3.

4. # Podział danych na zbiór treningowy i testowy

5. library(caret)

6. trainIndex <- createDataPartition(grzyby$class, p = 0.75, list = FALSE, times = 1)

7. sTrain <- grzyby[trainIndex,]

8. sTest <- grzyby[-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. d\_tree <- rpart(class ~ ., data = sTrain, method = "class", control = dt\_control, minsplit = 5)

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(d\_tree)

Classification tree:

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

minsplit = 5)

Variables actually used in tree construction:

[1] cap.color habitat odor population

Root node error: 2937/6093 = 0.48203

n= 6093

CP nsplit rel error xerror xstd

1 0.9720804 0 1.0000000 1.0000000 0.0132801

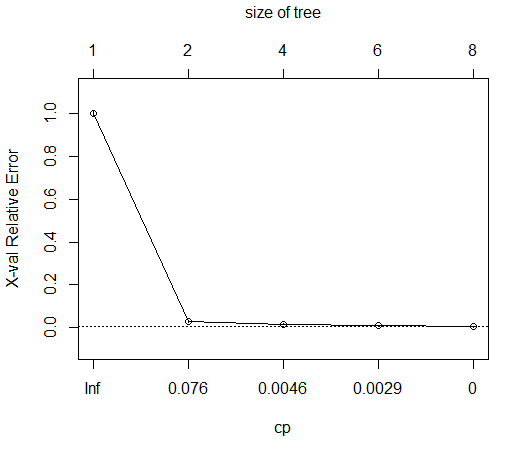
2 0.0059585 1 0.0279196 0.0279196 0.0030624

3 0.0035751 3 0.0160027 0.0160027 0.0023252

4 0.0023834 5 0.0088526 0.0088526 0.0017324

5 0.0000000 7 0.0040858 0.0040858 0.0011783

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:

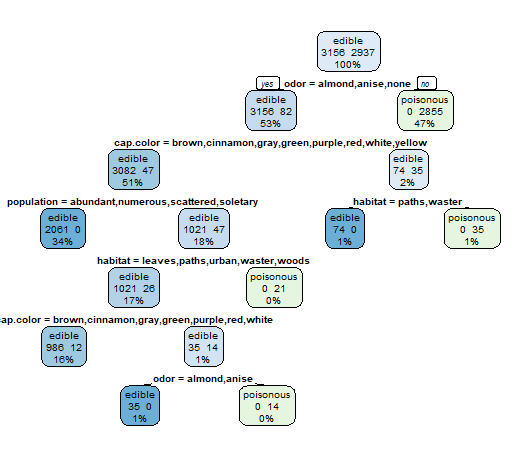
1. # Przycinanie drzewa na podstawie optymalnego cp

2. d\_tree\_pruned <- prune(d\_tree, cp = 0.002)

Po przycięciu drzewa możemy go narysować za pomocą kodu:

1. rpart.plot(d\_tree\_pruned, extra = 101, fallen.leaves = FALSE, tweak = 1.2, varlen = 10, faclen = 10)

Poniżej przedstawiono przycięte drzewo dla wybranych wartości cp=0.002.

****

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

1. # Predykcja na zbiorze testowym

2. d\_tree\_predict <- predict(d\_tree\_pruned, sTest, type = "class")

3.

4. # Macierz konfuzji i ocena modelu

5. library(caret)

6. confusionMatrix(d\_tree\_predict, sTest$class)

Poniżej przedstawiono wyniki dla wcześniej narysowanego drzewa.

> # Wyniki testów

> confusionMatrix(d\_tree\_predict, sTest$class)

Confusion Matrix and Statistics

Reference

Prediction edible poisonous

edible 1052 12

poisonous 0 967

Accuracy : 0.9941

95% CI : (0.9897, 0.9969)

No Information Rate : 0.518

P-Value [Acc > NIR] : < 2.2e-16

Kappa : 0.9882

Mcnemar's Test P-Value : 0.001496

Sensitivity : 1.0000

Specificity : 0.9877

Pos Pred Value : 0.9887

Neg Pred Value : 1.0000

Prevalence : 0.5180

Detection Rate : 0.5180

Detection Prevalence : 0.5239

Balanced Accuracy : 0.9939

'Positive' Class : edible

**7.** Metryka Accuracy dla przyciętego drzewa wynosi 0,9941 co wydaje się być bardzo dobrą wartością.