**Sztuczna Inteligencja 2020/21**

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

**Błażej Bałdyga, 02.06.2021**

Do uczenia maszynowego metodą drzew decyzyjnych wybrano bazę **Irys**, która zawiera dane dotyczące **kwiatów irysów**. Celem zadania jest klasyfikacja obserwacji względem kryterium **gatunku kwiatu**. Zbiór danych zawiera **150** próbek (obserwacji). Każda próbka zawiera **5** cech (tj. zmiennych przewidujących). Zmienna wyjaśniana (zależna, przewidywana) o nazwie **Classification** posiada **trzy** różne wartości: **Setosa, Versicolor i Virginica** stanowiące kryterium klasyfikacji. Wartości tej zmiennej mają następującą interpretację **jakiego gatunku jest dany kwiat**

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

|  |  |  |
| --- | --- | --- |
| Lp. | Nazwa Cechy | Opis |
| 1 | **Długość kielicha** | Ile wynosi długość kielicha w cm |
| 2 | **Szerokość kielicha** | Ile wynosi szerokość kielicha w cm |
| 3 | **Długość płatków** | Ile wynosi długość płatków kwiatka w cm |
| 4 | **Szerokość płatków** | Ile wynosi szerokość płatków kwiatka w cm |

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

iris ←read.csv("/Users/blazejbaldyga/Downloads/DrzewaDecyzyjne/iris.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:

'data.frame': 150 obs. of 5 variables:

$ SL : num 5 7.9 5.8 5.7 5.4 5.2 6.4 5.5 4.4 5 ...

$ SW : num 3.5 3.8 2.8 2.6 3.9 2.7 2.7 2.6 2.9 3.3 ...

$ PL : num 1.3 6.4 5.1 3.5 1.7 3.9 5.3 4.4 1.4 1.4 ...

$ PW : num 0.3 2 2.4 1 0.4 1.4 1.9 1.2 0.2 0.2 ...

$ Classification: chr "Setosa" "Virginica" "Virginica" "Versicolor" ...

W związku z tym musimy zmienić typ zmiennych: **Classification** na obsługiwany typ za pomocą kodu:

**iris <- mutate\_at(iris,vars(Classification),list(as.factor))**

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

'data.frame': 150 obs. of 5 variables:

$ SL : num 5 7.9 5.8 5.7 5.4 5.2 6.4 5.5 4.4 5 ...

$ SW : num 3.5 3.8 2.8 2.6 3.9 2.7 2.7 2.6 2.9 3.3 ...

$ PL : num 1.3 6.4 5.1 3.5 1.7 3.9 5.3 4.4 1.4 1.4 ...

$ PW : num 0.3 2 2.4 1 0.4 1.4 1.9 1.2 0.2 0.2 ...

$ Classification: Factor w/ 3 levels "Setosa","Versicolor",..: 1 3 3 2 1 2 3 2 1 1 ...

**3.** Nasze dane zawierają następującą liczbę próbek każdej z klas: **50** 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.

**set.seed(1456)**

**trainIndex <- createDataPartition(iris$Classification,p = 0.75, list = FALSE, times = 1)**

**sTrain <- iris[trainIndex,]**

**sTest <- iris[-trainIndex,]**

**count(sTrain,Classification)**

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

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

**d\_tree <- rpart(Classification~., data = sTrain, method = "class", control = dt\_control, minsplit = 5)**

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

**4.** 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).

Classification tree:

rpart(formula = Classification ~ ., data = sTrain, method = "class", control = dt\_control, minsplit = 5)

Variables actually used in tree construction:

[1] PL PW

Root node error: 76/114 = 0.66667

n= 114

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | CP | nsplit | Rel error | xerror | xstd |
| 1 | 0.50000 | 0 | 1.000000 | 1.15789 | 0.058947 |
| 2 | 0.46053 | 1 | 0.500000 | 0.59211 | 0.068670 |
| 3 | 0.00000 | 2 | 0.039474 | 0.11842 | 0.037883 |

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:

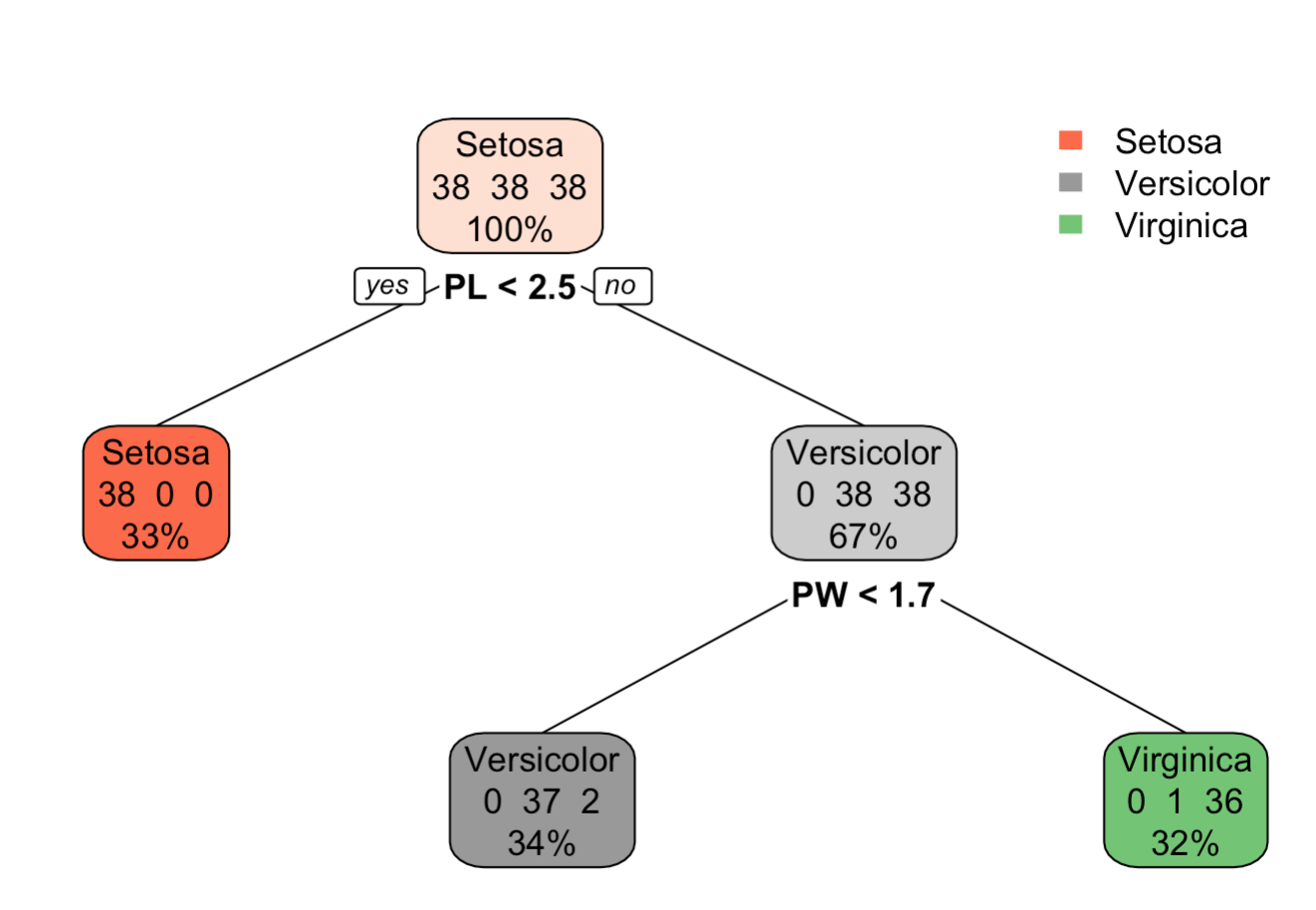
**dl\_tree <- prune(d\_tree, cp = 0)**

Po przycięciu drzewa możemy go narysować za pomocą kodu:

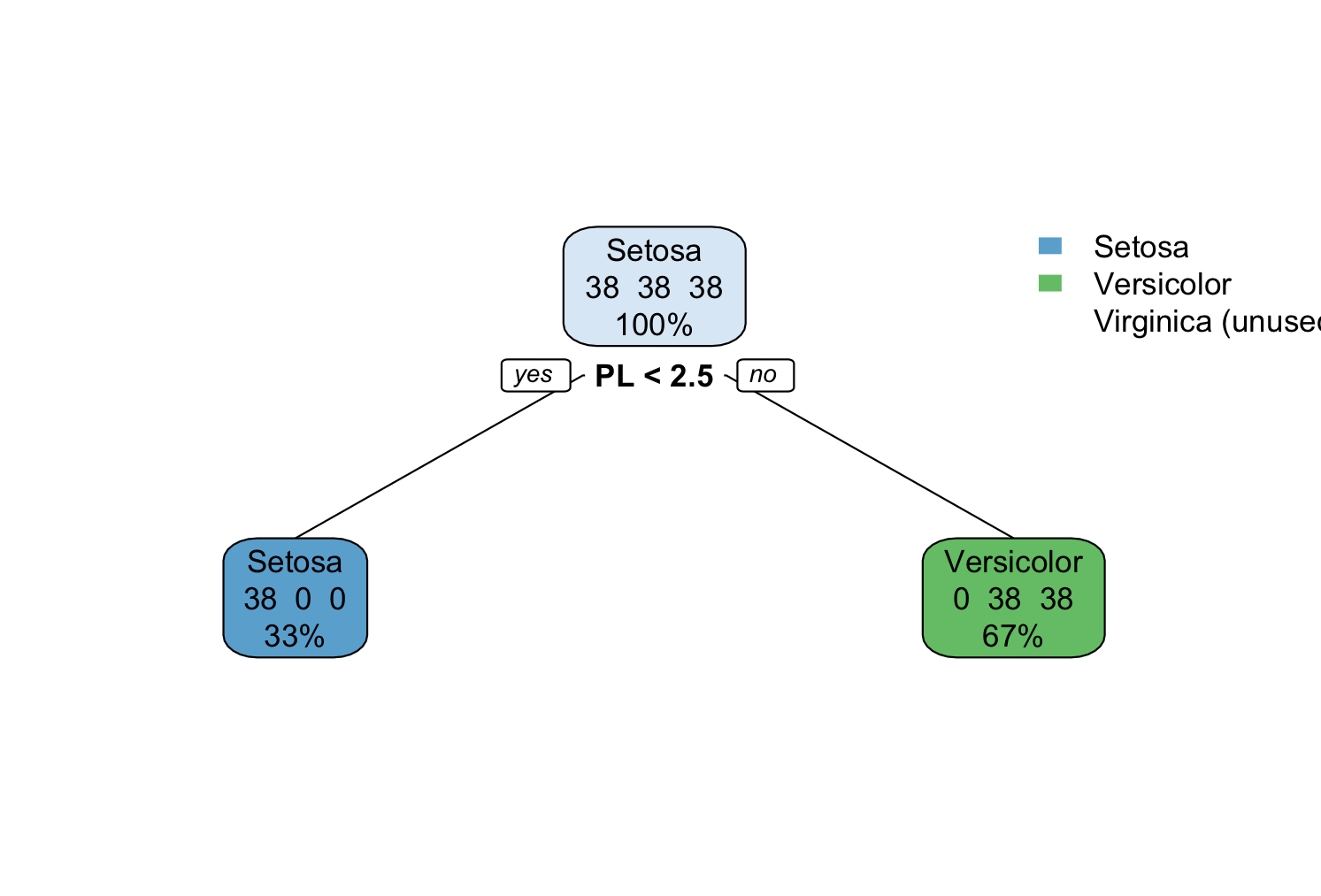
**rpart.plot(dl\_tree, extra = 101, fallen.leaves = FALSE, tweak = 1.0, varlen = 10, faclen = 10)**

Poniżej przedstawiono przycięte drzewa dla wybranych wartości cp.

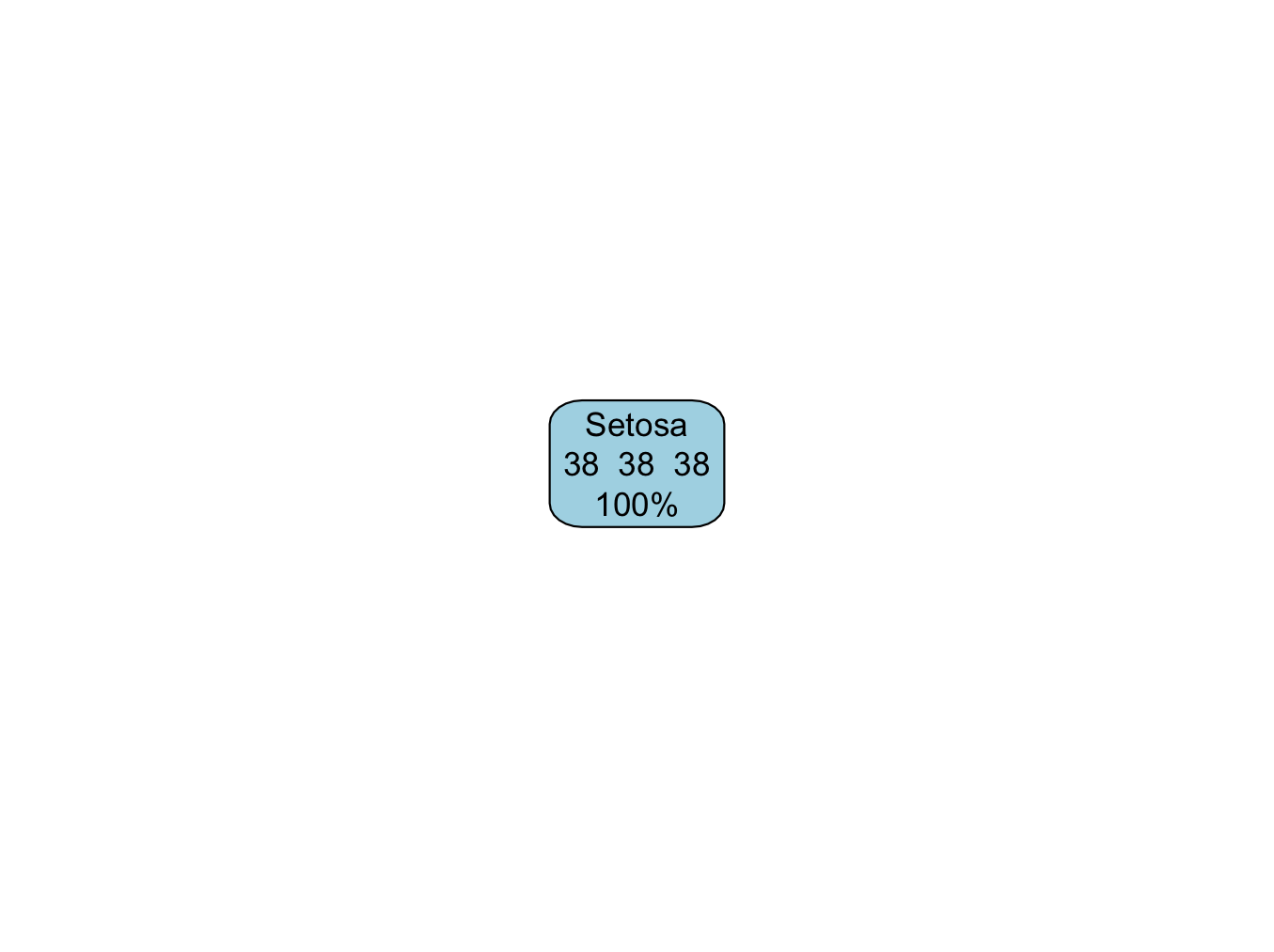
Rysunek 1 cp = 0



**Rysunek 2 cp = 0.46053**



**Rysunek 3 cp = 0.50000**



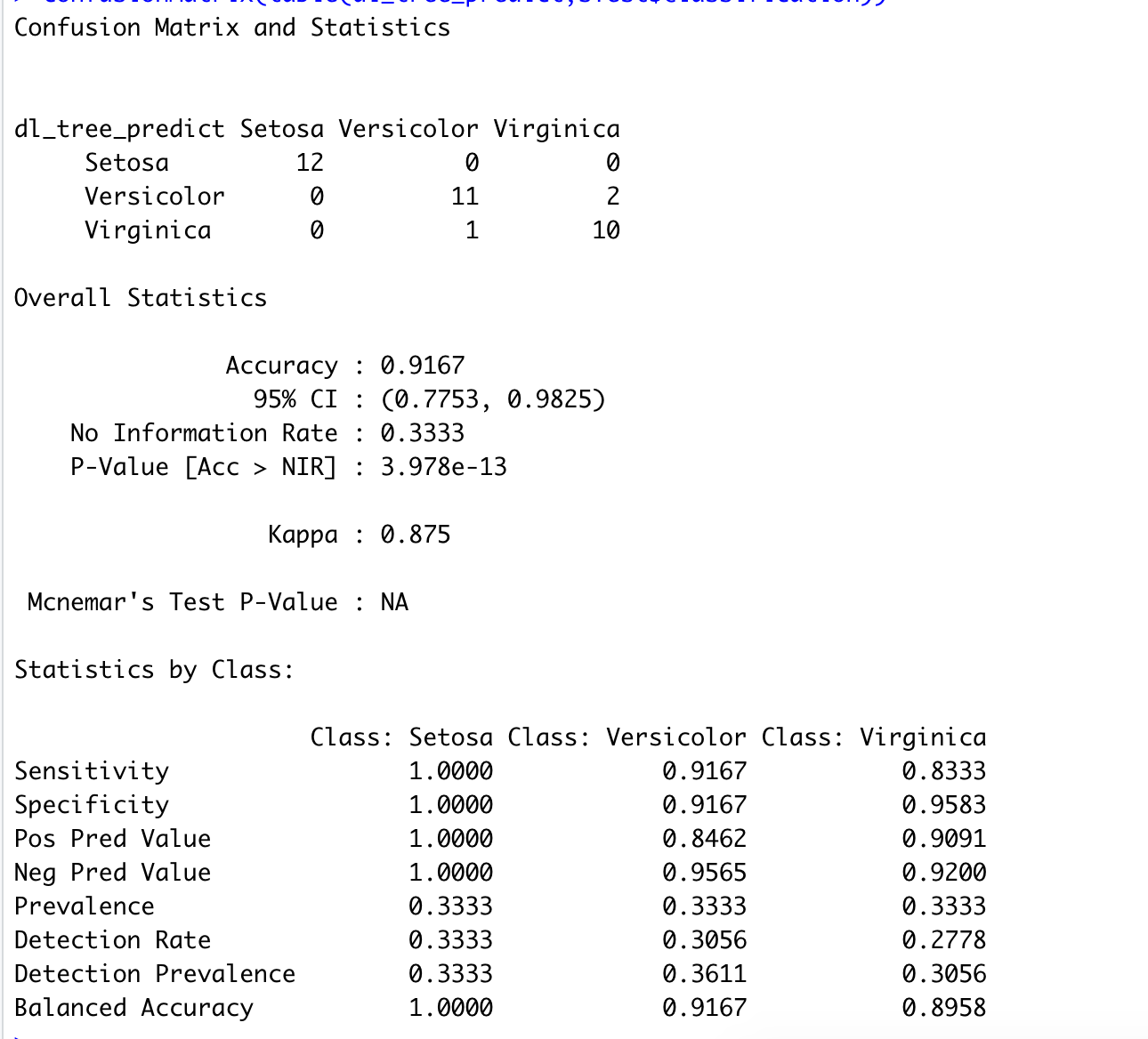
**5**. Na zakończenie należy przetestować nasze drzewa wykorzystując zbiór testowy. Robimy to za pomocą kodu:

**dl\_tree\_predict <- predict(dl\_tree, sTest, type="class")**

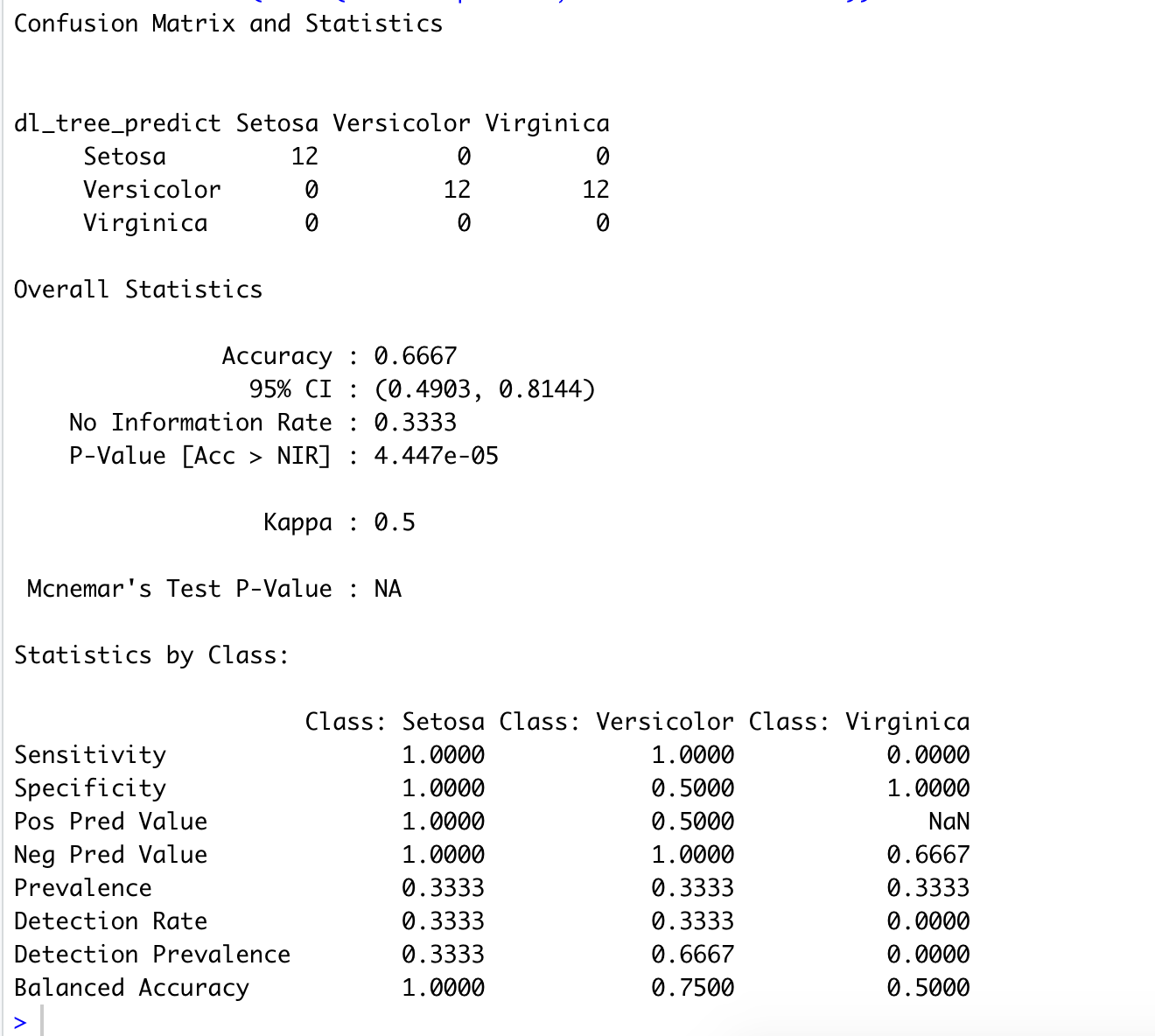
**confusionMatrix(table(dl\_tree\_predict,sTest$Classification))**

Poniżej przedstawiono wyniki testów dla wcześniej narysowanych drzew.

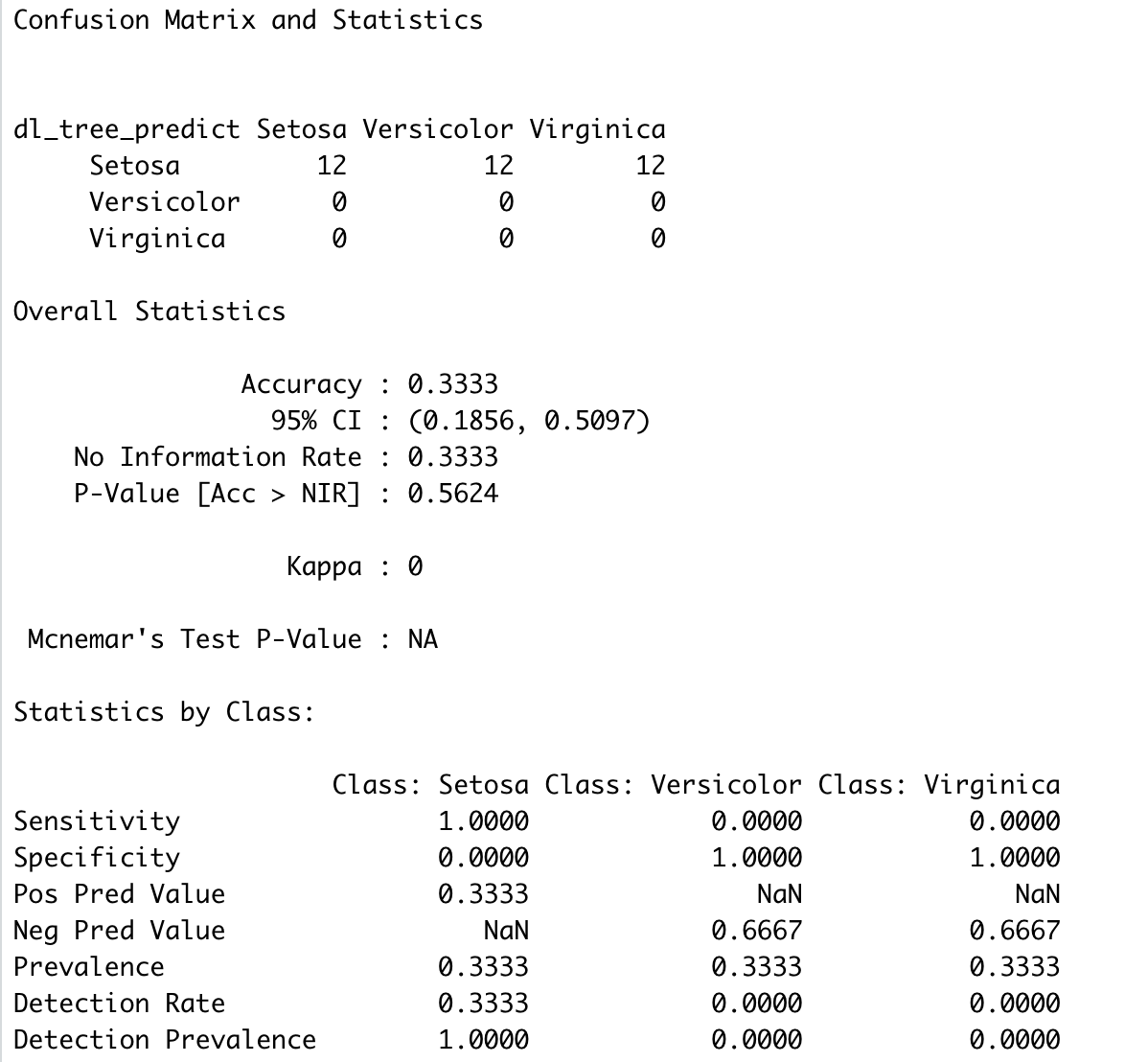
**Wyniki dla rysunku 1**



**Wyniki dla rysunku 2**



**Wyniki dla rysunku 3**



6. Porównując wartości metryki **Accuracy** można przyjąć, że najlepsze drzewo zostało przedstawione na rysunku **1**, ma 2 węzłów (nspit), oraz wartość cp = 0.