Drzewa klasyfikacyjne i regresyjne

Opis teorii

Drzewo decyzyjne jest jednym z szeroko stosowanych algorytmów w uczeniu maszynowym, zapewniającym solidną podstawę dla innych podejść. Podstawowym celem korzystania z drzewa decyzyjnego jest stworzenie modelu, który może przewidzieć docelową klasę lub wartość zmiennej poprzez naukę prostych reguł podejmowania decyzji wywnioskowanych z wcześniejszych danych (danych szkoleniowych). Używa wykresu przypominającego drzewo, aby pokazać prognozy wynikające z serii podziałów na podstawie cech.

Jednym ze sposobów myślenia o drzewie decyzyjnym jest seria węzłów lub wykres kierunkowy, który zaczyna się od pojedynczego węzła u podstawy i rozciąga się na wiele węzłów liści reprezentujących kategorie, które drzewo może klasyfikować. Każdy węzeł w drzewie określa test dla danego atrybutu. Każda gałąź wychodząca z węzła odpowiada jednej z możliwych wartości atrybutu. Każdy węzeł na liściu przypisuje wartość przewidywaną.

Formalnie drzewo decyzyjne jest **acyklicznym spójnym grafem skierowanym**. Każdemu jego węzłowi, będącemu liściem, przyporządkowane jest w przypadku drzew klasyfikacyjnych oznaczenie klasy, a w przypadku drzew regresyjnych wartość numeryczna, a każdej z gałęzi reguła decyzyjna, czyli warunek odnoszący się do wartości zmiennych w zbiorze wejściowym i mówiący, w jakim przypadku należy pójść daną gałęzią.

Drzewo decyzyjne działa w oparciu o prostą zasadę **iteracyjnych podziałów**: na każdym poziomie drzewa dane są dzielone na podgrupy według określonego kryterium, takiego jak entropia, wskaźnik Gini lub zmniejszenie wariancji. Proces budowy drzewa rozpoczyna się od węzła głównego (korzenia), który obejmuje wszystkie dane wejściowe. Następnie algorytm wybiera cechę, która najlepiej różnicuje dane pod kątem zmiennej docelowej, i przeprowadza podział. Ten proces jest powtarzany dla każdego węzła, aż do spełnienia kryteriów zatrzymania, takich jak:

 Osiągnięcie maksymalnej głębokości drzewa.

 Brak wystarczającej liczby danych w węźle do dalszego podziału.

 Brak dalszej poprawy jakości podziału.

Podstawowe elementy konstrukcyjne drzewa decyzyjnego obejmują:

- **Węzły wewnętrzne (internal nodes):** odpowiadające testom decyzyjnym dla cech.
- **Gałęzie (branches):** reprezentujące wyniki testów prowadzących do kolejnych węzłów.
- **Liście (leaves):** końcowe węzły, przypisujące przewidywaną wartość lub klasę.

Drzewa klasyfikacyjne i regresyjne

Drzewa decyzyjne dzielą się na dwie podstawowe kategorie:

 Drzewa klasyfikacyjne:

- Stosowane, gdy zmienna docelowa jest kategoryczna.
- Do oceny podziałów wykorzystują miary takie jak wskaźnik Gini lub entropia.
- Każdy liść przypisuje jedną z klas na podstawie dominujących etykiet w grupie danych końcowych.

 Drzewa regresyjne:

- Wykorzystywane, gdy zmienna docelowa jest ciągła.
- Podziały są oceniane na podstawie zmniejszenia wariancji lub sumy kwadratów reszt.
- Liście przewidują wartości numeryczne, zwykle na podstawie średniej lub mediany obserwacji w grupie.

Zalety

- **Prostota wyników.** W większości przypadków interpretacja wyników w postaci drzewa jest bardzo prosta. Ta prostota jest cenna nie tylko dlatego, że nowe przypadki są szybko klasyfikowane (łatwiej sprawdzić kilka warunków logicznych niż obliczać jakąś statystykę klasyfikacyjną dla każdej możliwej grupy lub przewidywanej wartości na podstawie wartości predyktorów w modelu wykorzystującym skomplikowane równania nieliniowe), lecz także z powodu znacznie prostszego "modelu" wyjaśniającego dlaczego obserwacje są klasyfikowane lub przewidywane w taki sposób a nie inny (np. w analizie problemów biznesowych łatwiej przedstawić kilka warunków *jeżeli-to* niż jakieś skomplikowane równania).

- **Metody drzew decyzyjnych są nieparametryczne i nieliniowe.** Drzewo decyzyjne jest nieparametrycznym algorytmem, w przeciwieństwie np. do regresji logistycznej czy sieci neuronowych, które przetwarzają dane wejściowe przekształcone w tensor, używając dużej liczby współczynników (zwanych parametrami), poprzez mnożenie tensorów. Wyniki wykorzystujące metody drzew klasyfikacyjnych i regresyjnych dają się ująć w postaci kilku (zazwyczaj niewielu) warunków logicznych typu *jeżeli-to* (z węzłów drzewa). Nie ma więc na wstępie żadnego założenia, co do natury związku pomiędzy predyktorami a zmienną zależną - czy jest on liniowy, czy też związek ten modeluje konkretna funkcja wiążąca, ani nawet czy jest to zależność monotoniczna. Metody drzew klasyfikacyjnych nadają się więc dobrze do zadań data mining, gdzie często wiedza *a priori* jest bardzo mała, i nie ma żadnych rozsądnych teorii lub ocen, które zmienne są ze sobą powiązane i w jaki sposób. W tego rodzaju analizach metody drzew klasyfikacyjnych potrafią wykryć związki pomiędzy kilkoma zmiennymi, które nie zostałyby wykryte przez inne techniki analityczne.

- **Moc wyjaśniająca** - łatwe do wyjaśnienia i zinterpretowania, dane wyjściowe drzew decyzyjnych są łatwe do interpretacji. Może być zrozumiany przez każdego bez wiedzy analitycznej, matematycznej lub statystycznej.
- **Eksploracyjna analiza danych** - drzewa decyzyjne pozwalają analitykom szybko zidentyfikować istotne zmienne i istotne relacje między dwiema lub więcej zmiennymi, pomagając w ten sposób ujawnić sygnał, który zawiera wiele zmiennych wejściowych.
- **Minimalne czyszczenie danych** - ponieważ drzewa decyzyjne są odporne na wartości odstające i brakujące wartości, wymagają mniej czyszczenia danych niż inne algorytmy.
- **Wszystkie typy danych** - drzewa decyzyjne mogą dokonywać klasyfikacji na podstawie zarówno zmiennych numerycznych, jak i kategorycznych.

Wady

- **Przeuczenie** - częstym błędem w drzewach decyzyjnych jest nadmierne dopasowanie. Dwa sposoby regulowania drzewa decyzyjnego to ustawienie ograniczeń parametrów modelu i uproszczenie modelu poprzez przycinanie

Główne różnice między drzewami klasyfikacyjnymi, a regresyjnymi

Cecha	Drzewo klasyfikacyjne	Drzewo regresyjne
Cel	Przewiduje kategorie lub klasy.	Przewiduje wartości ciągłe.
Zmienne docelowe	Zmienne kategoryczne (np. tak/nie, klasa 1/klasa 2).	Zmienne numeryczne (np. cena, dochód, temperatura).
Funkcje podziału	Wskaźnik Gini, entropia, informacja wzajemna.	Zmniejszenie błędu średniokwadratowego (MSE), zmniejszenie wariancji.
Liście drzewa	Każdy liść przypisuje konkretną klasę.	Każdy liść przypisuje wartość liczbową (średnią/medianę).
Wynik	Klasyfikacja danych wejściowych.	Estymacja wartości liczbowych.
Interpretacja	Wynik w postaci procentowej przynależności do danej klasy.	Wynik jako konkretna wartość liczbową dla zmiennej zależnej.
Przykłady zastosowań	Diagnoza chorób, analiza zdolności kredytowej klientów, wykrywanie oszustw.	Przewidywanie cen nieruchomości, prognoza sprzedaży, analiza zysków.

Analiza algorytmu drzewa klasyfikacyjnego

Do analizy algorytmu drzewa klasyfikacyjnego został wykorzystnay dataset „heart.csv" zasięgnięty z kaggle.

Dane zawierają następujące informacje:

- Wiek: wiek pacjenta [lata]
- Sex: płeć pacjenta [M: mężczyzna, F: kobieta].
- ChestPainType: typ bólu w klatce piersiowej [TA: Typowa dławica piersiowa, ATA: Nietypowa dławica piersiowa, NAP: NonAnginal Pain, ASY: Asymptomatic]
- RestingBP: spoczynkowe ciśnienie krwi [mm Hg]
- Cholesterol: cholesterol w surowicy [mm/dl]
- FastingBS: poziom cukru we krwi na czczo [1: jeśli FastingBS > 120 mg/dl, 0: w przeciwnym razie]
- RestingECG: wyniki elektrokardiogramu spoczynkowego [Normal: normalny, ST: nieprawidłowości załamka ST: T (odwrócenie załamka T i/lub uniesienie lub obniżenie odcinka ST o > 0,05 mV), LVH: prawdopodobny lub definitywny przerost lewej komory według kryteriów Estesa].
- MaxHR: osiągnięta maksymalna częstość akcji serca [wartość liczbowa między 60 a 202].
- ExerciseAngina: wysiłkowa dławica piersiowa [Y: Tak, N: Nie]
- Oldpeak: oldpeak = ST [Wartość liczbowa mierzona w depresji]
- ST_Slope: nachylenie szczytowego wysiłkowego odcinka ST [Up: upsloping, Flat: flat, Down: downsloping]
- HeartDisease: klasa wyjściowa [1: choroba serca, 0: normalny] (zmienna celu)

Biblioteki, jakie zostały wykorzystane

```
# Wczytanie bibliotek
library(tidyverse)
library(caret)
library(rpart)
library(rpart.plot)
library(randomForest)
library(pROC)
library(ggplot2)
library(dplyr)
```

Wczytanie danych

```
# Wczytanie danych
data <- read.csv("heart.csv")

param_grid <- expand.grid(
  maxdepth = c(3, 5, 7, 9, 11),
  minsplit = c(2, 5, 10, 20),
  cp = seq(0.001, 0.1, by = 0.01)
)
```

Przekształcenie danych kategorycznych na faktory

```
# Przekształcenie danych kategorycznych na faktory
data$Sex <- as.factor(data$Sex)
data$ChestPainType <- as.factor(data$ChestPainType)
data$RestingECG <- as.factor(data$RestingECG)
data$ExerciseAngina <- as.factor(data$ExerciseAngina)
data$ST_Slope <- as.factor(data$ST_Slope)
data$HeartDisease <- as.factor(data$HeartDisease)
```

Podział na zbiory treningowy i testowy 70% trening, 30% test)

```
# Podział na zbiory treningowy i testowy (70% trening, 30% test)
set.seed(123)
trainIndex <- createDataPartition(data$HeartDisease, p = 0.7, list = FALSE)
trainData <- data[trainIndex, ]
testData <- data[-trainIndex, ]
```

Funkcja do trenowania modelu i obliczania metryk z użyciem cross-walidacji

```
# Funkcja do trenowania modelu i obliczania metryk z użyciem cross-walidacji
evaluate_model_cv <- function(params, trainData, folds = 10) {
  # Przygotowanie zbiorów do cross-walidacji
  cv_folds <- createFolds(trainData$HeartDisease, k = folds, list = TRUE)

  # Miejsce na metryki
  metrics <- data.frame(Accuracy = numeric(), AUC = numeric(), Precision = numeric(), Recall = numeri
c())

  # Cross-walidacja
  for (fold in cv_folds) {
    train_fold <- trainData[-fold, ]
    test_fold <- trainData[fold, ]

    # Trenowanie modelu
    model <- rpart(
      HeartDisease ~ .,
      data = train_fold,
      method = "class",
```

```
      control = rpart.control(
        maxdepth = as.numeric(params["maxdepth"]),
        minsplit = as.numeric(params["minsplit"]),
        cp = as.numeric(params["cp"])
      )
    )

    # Predykcja na zbiorze testowym
    predictions <- predict(model, newdata = test_fold, type = "class")
    probabilities <- predict(model, newdata = test_fold, type = "prob")[, 2]

    # Obliczanie metryk
    accuracy <- mean(predictions == test_fold$HeartDisease)
    roc_curve <- roc(test_fold$HeartDisease, probabilities, levels = levels(test_fold$HeartDisease))
    auc_score <- auc(roc_curve)
    cm <- confusionMatrix(predictions, test_fold$HeartDisease)
    precision <- cm$byClass["Precision"]
    recall <- cm$byClass["Recall"]

    # Zapis metryk
    metrics <- rbind(metrics, c(Accuracy = accuracy, AUC = auc_score, Precision = precision, Recall =
recall))
  }

  # Średnie metryki z CV
  colMeans(metrics, na.rm = TRUE)
}
```

Zastosowanie funkcji dla całej siatki parametrów i wyświetlenie wyników

```
# Zastosowanie funkcji dla całej siatki parametrów results <- param_grid metric_results <-
t(apply(param_grid, 1, function(row) evaluate_model_cv(as.list(row), trainData)))

# Poprawienie nazw kolumn w wynikowych metrykach colnames(metric_results)
<- c("Accuracy", "AUC", "Precision", "Recall") results <- cbind(results,
metric_results)

# Wyświetlenie wyników print(results)
```

```
    maxdepth minsplit    cp  Accuracy       AUC Precision    Recall 1
3          2 0.001 0.8130414 0.8526440 0.7896001 0.8045567
2          5        2 0.001 0.7916487 0.8562297 0.7636179 0.7833744
3          7        2 0.001 0.8007112 0.8072871 0.7804433 0.7810345
4          9        2 0.001 0.7666239 0.7640798 0.7511358 0.7453202
5          11        2 0.001 0.7777305 0.7675345 0.7474583 0.7596059
6          3        5 0.001 0.8086935 0.8674390 0.7809525 0.8081281
7          5        5 0.001 0.8056384 0.8702143 0.7743817 0.8049261
8          7        5 0.001 0.8087927 0.8270623 0.8029633 0.7666256
9          9        5 0.001 0.7839347 0.8049455 0.7695356 0.7378079
10         11        5 0.001 0.7932284 0.7902638 0.7787168 0.7599754
11         3       10 0.001 0.8195715 0.8744598 0.7937326 0.8153941
12         5       10 0.001 0.7948226 0.8536142 0.7719433 0.7738916
13         7       10 0.001 0.7946791 0.8526720 0.7882432 0.7500000
14         9       10 0.001 0.7963759 0.8492530 0.7701450 0.7805419
15         11       10 0.001 0.7776137 0.8588338 0.7687531 0.7320197
16         3       20 0.001 0.8086920 0.8607835 0.7797206 0.8044335
17         5       20 0.001 0.7852408 0.8572582 0.7640190 0.7588670
18         7       20 0.001 0.7932494 0.8768120 0.7900086 0.7394089
19         9       20 0.001 0.7792067 0.8740456 0.7834956 0.7041872
```

```
20          11       20 0.001 0.7868395 0.8626206 0.7787970 0.7317734
21          3        2 0.011 0.8102885 0.8614597 0.7796074 0.8051724
22          5        2 0.011 0.8196219 0.8637507 0.8126795 0.7773399
23          7        2 0.011 0.8009013 0.8703059 0.7826195 0.7764778
24          9        2 0.011 0.8136222 0.8706434 0.8102069 0.7634236
25          11        2 0.011 0.8055529 0.8686624 0.7930817 0.7633005
26          3        5 0.011 0.8132345 0.8618839 0.7803971 0.8109606
27          5        5 0.011 0.8116132 0.8734821 0.7922010 0.7944581
28          7        5 0.011 0.7960470 0.8451581 0.7911809 0.7460591
29          9        5 0.011 0.8226442 0.8872260 0.8049881 0.8019704
30          11        5 0.011 0.8118906 0.8721459 0.7981721 0.7774631
31          3       10 0.011 0.8210726 0.8770880 0.7907173 0.8184729
32          5       10 0.011 0.8070574 0.8657837 0.7975027 0.7656404
33          7       10 0.011 0.7982547 0.8604352 0.7845008 0.7532020
34          9       10 0.011 0.8135508 0.8712669 0.7993274 0.7842365
35          11       10 0.011 0.8055933 0.8689698 0.8089276 0.7392857
36          3       20 0.011 0.8055529 0.8599384 0.7696116 0.8080049
37          5       20 0.011 0.8163339 0.8809014 0.8048706 0.7843596
38          7       20 0.011 0.8133894 0.8826084 0.7985454 0.7769704
39          9       20 0.011 0.8149519 0.8771682 0.8007226 0.7839901
40          11       20 0.011 0.8025721 0.8758395 0.8008851 0.7488916
41          3        2 0.021 0.7978648 0.8372771 0.7941325 0.7529557
42          5        2 0.021 0.7961298 0.8405395 0.7889577 0.7485222
43          7        2 0.021 0.7946303 0.8339646 0.7843367 0.7598522
44          9        2 0.021 0.7933478 0.8432888 0.7785651 0.7571429
45          11        2 0.021 0.8055735 0.8588627 0.7984042 0.7726601
46          3        5 0.021 0.7976129 0.8456977 0.7796375 0.7697044
47          5        5 0.021 0.7992907 0.8604894 0.7848948 0.7699507
48          7        5 0.021 0.7931639 0.8418256 0.7870089 0.7529557
49          9        5 0.021 0.8088462 0.8604659 0.8020884 0.7735222
50          11        5 0.021 0.8087176 0.8629391 0.7945290 0.7842365
51          3       10 0.021 0.7960577 0.8617833 0.7629487 0.7871921
52          5       10 0.021 0.8085733 0.8604484 0.7926645 0.7727833
53          7       10 0.021 0.7898691 0.8376063 0.7702419 0.7671182
54          9       10 0.021 0.8055422 0.8534336 0.7851165 0.7799261
55          11       10 0.021 0.8023916 0.8494160 0.7948905 0.7493842
56          3       20 0.021 0.8070822 0.8553203 0.7962216 0.7705665
57          5       20 0.021 0.8027755 0.8640497 0.7764803 0.7847291
58          7       20 0.021 0.8086455 0.8522038 0.7890987 0.7879310
59          9       20 0.021 0.8020662 0.8594471 0.7811972 0.7763547
60          11       20 0.021 0.8148615 0.8690006 0.7980520 0.7902709
61          3        2 0.031 0.7885012 0.8166279 0.8051710 0.7177340
62          5        2 0.031 0.7823832 0.8073394 0.8023006 0.6896552
63          7        2 0.031 0.7977595 0.8095686 0.8196488 0.7073892
64          9        2 0.031 0.7897962 0.8163148 0.7961282 0.7237685
65          11        2 0.031 0.7854724 0.8179915 0.7966495 0.7145320
66          3        5 0.031 0.7866228 0.8105862 0.7925883 0.7107143
67          5        5 0.031 0.7915865 0.8031192 0.7934072 0.7277094
68          7        5 0.031 0.7978152 0.8132056 0.8262529 0.6971675
69          9        5 0.031 0.8023195 0.8123733 0.8312300 0.6998768
70          11        5 0.031 0.7929773 0.8142927 0.8139550 0.7034483
71          3       10 0.031 0.7868132 0.7948508 0.7926150 0.7215517
72          5       10 0.031 0.7883665 0.7998978 0.8103027 0.7034483
73          7       10 0.031 0.7886348 0.7976515 0.8098488 0.6970443
74          9       10 0.031 0.7886928 0.8028022 0.7947284 0.7179803
75          11       10 0.031 0.7915388 0.8183012 0.8217223 0.6891626
76          3       20 0.031 0.7934711 0.8018152 0.8089041 0.7248768
77          5       20 0.031 0.7698798 0.8022342 0.7623290 0.7178571
78          7       20 0.031 0.7979808 0.8056985 0.8200055 0.7113300
79          9       20 0.031 0.7869471 0.8047746 0.8125535 0.6895320
80          11       20 0.031 0.7901026 0.7888480 0.8110826 0.6970443
81          3        2 0.041 0.7915652 0.7898810 0.7947525 0.7247537
82          5        2 0.041 0.7868990 0.7887011 0.7890685 0.7214286
83          7        2 0.041 0.7949931 0.7911210 0.7934103 0.7355911
84          9        2 0.041 0.7883284 0.7899973 0.7921215 0.7214286
85          11        2 0.041 0.7888278 0.7907567 0.7913892 0.7216749
86          3        5 0.041 0.7974176 0.7913699 0.7998235 0.7342365
```

```
87          5        5 0.041 0.7979964 0.7924877 0.7957153 0.7391626
88          7        5 0.041 0.7915045 0.7885263 0.7883179 0.7350985
89          9        5 0.041 0.7946875 0.7888725 0.7909426 0.7418719
90         11        5 0.041 0.7900240 0.7887144 0.7872353 0.7322660
91          3       10 0.041 0.7929090 0.7903400 0.7977439 0.7315271
92          5       10 0.041 0.7914061 0.7885670 0.7875951 0.7348522
93          7       10 0.041 0.7978606 0.7922578 0.7998671 0.7346059
94          9       10 0.041 0.7930308 0.7905049 0.7948876 0.7315271
95         11       10 0.041 0.7961077 0.7921319 0.7998188 0.7282020
96          3       20 0.041 0.7945807 0.7933381 0.8079281 0.7210591
97          5       20 0.041 0.7915419 0.7877634 0.7814551 0.7384236
98          7       20 0.041 0.7962874 0.7902887 0.7924156 0.7418719
99          9       20 0.041 0.7932021 0.7912397 0.7953152 0.7286946
100        11       20 0.041 0.7931277 0.7913150 0.7962854 0.7243842
101         3        2 0.051 0.7918750 0.7900674 0.7915470 0.7320197
102         5        2 0.051 0.7945444 0.7906548 0.7905456 0.7349754
103         7        2 0.051 0.7883997 0.7884975 0.7861736 0.7284483
104         9        2 0.051 0.7947115 0.7889231 0.7888961 0.7419951
105        11        2 0.051 0.7946280 0.7918856 0.7938375 0.7284483
106         3        5 0.051 0.7933616 0.7883333 0.7916986 0.7428571
107         5        5 0.051 0.7932128 0.7888478 0.7902570 0.7384236
108         7        5 0.051 0.7899279 0.7906856 0.7922991 0.7252463
109         9        5 0.051 0.7917953 0.7912192 0.7952270 0.7254926
110        11        5 0.051 0.7934470 0.7925661 0.8037328 0.7219212
111         3       10 0.051 0.7947638 0.7904433 0.7917738 0.7391626
112         5       10 0.051 0.7898661 0.7888714 0.7893069 0.7278325
113         7       10 0.051 0.7915037 0.7886631 0.7874485 0.7349754
114         9       10 0.051 0.7946154 0.7914874 0.7982503 0.7316502
115        11       10 0.051 0.7915285 0.7894650 0.7929150 0.7317734
116         3       20 0.051 0.7976229 0.7929376 0.7993624 0.7310345
117         5       20 0.051 0.7914183 0.7905515 0.7919587 0.7250000
118         7       20 0.051 0.7869723 0.7893439 0.7891822 0.7176108
119         9       20 0.051 0.7883425 0.7887791 0.7859720 0.7237685
120        11       20 0.051 0.7918109 0.7919978 0.8006656 0.7215517
121         3        2 0.061 0.7885478 0.7882225 0.7847779 0.7323892
122         5        2 0.061 0.7883894 0.7779143 0.7890746 0.7242611
123         7        2 0.061 0.7934135 0.7883032 0.7843430 0.7422414
124         9        2 0.061 0.7931639 0.7881048 0.7861467 0.7422414
125        11        2 0.061 0.7914904 0.7884250 0.7826977 0.7355911
126         3        5 0.061 0.7854724 0.7877487 0.7792477 0.7214286
127         5        5 0.061 0.7930926 0.7881226 0.7867068 0.7421182
128         7        5 0.061 0.7931639 0.7881623 0.7847472 0.7421182
129         9        5 0.061 0.7930903 0.7879119 0.7856917 0.7413793
130        11        5 0.061 0.7915037 0.7890900 0.7929562 0.7350985
131         3       10 0.061 0.7931609 0.7880569 0.7856341 0.7417488
132         5       10 0.061 0.7918719 0.7892915 0.7904876 0.7354680
133         7       10 0.061 0.7934066 0.7885892 0.7858833 0.7427340
134         9       10 0.061 0.7930804 0.7881404 0.7864404 0.7419951
135        11       10 0.061 0.7885997 0.7877354 0.7916499 0.7320197
136         3       20 0.061 0.7901126 0.7886631 0.7891489 0.7316502
137         5       20 0.061 0.7881819 0.7881248 0.7831794 0.7246305
138         7       20 0.061 0.7903056 0.7881396 0.7889745 0.7349754
139         9       20 0.061 0.7867682 0.7880282 0.7836100 0.7242611
140        11       20 0.061 0.7929918 0.7880651 0.7896776 0.7422414
141         3        2 0.071 0.7932097 0.7883511 0.7867812 0.7427340
142         5        2 0.071 0.7931128 0.7883032 0.7849655 0.7422414
 [ reached 'max' / getOption("max.print") -- omitted 58 rows ]
```

Najlepsze parametry na podstawie AUC (można zmienić na "Accuracy", "Precision", "Recall")

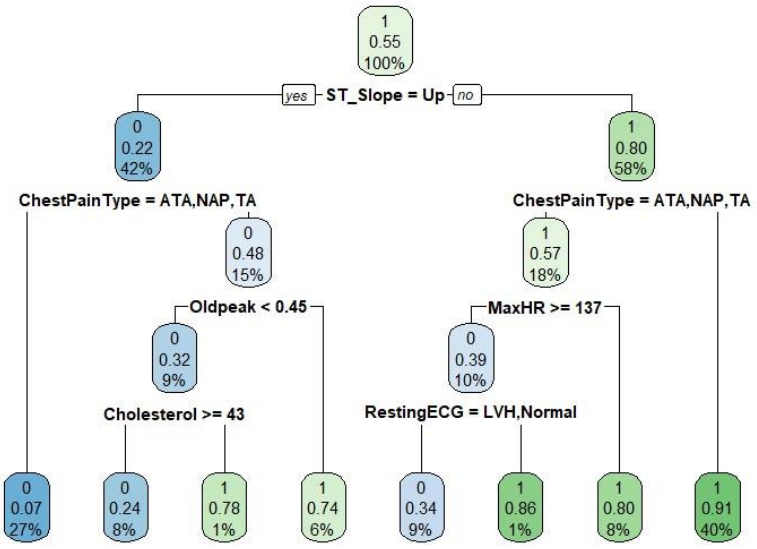
```
# Najlepsze parametry na podstawie AUC (można zmienić na "Accuracy", "Precision", "Recall")
best_params <- results[which.max(results$Accuracy), ]
print(best_params)

   maxdepth minsplit    cp  Accuracy      AUC Precision    Recall
29        9        5 0.011 0.8226442 0.887226 0.8049881 0.8019704
```

Trenowanie modelu z najlepszymi parametrami na całym zbiorze danych wraz z wizualizacją

```
# Trenowanie modelu z najlepszymi parametrami na całym zbiorze danych
best_model <- rpart(
  HeartDisease ~ .,
  data = trainData,
  method = "class",
  control = rpart.control(
    maxdepth = best_params$maxdepth,
    minsplit = best_params$minsplit,
    cp = best_params$cp
  )
)

# Wizualizacja najlepszego drzewa
rpart.plot(best_model)
```



Powyższe drzewo decyzyjne rozpoczyna klasyfikację od **warunku na zmiennej ST_Slope** Sprawdza, czy `ST_Slope = Up` (lewa gałąź) czy też jest różne od „Up” (prawa gałąź).

- **Jeżeli** `ST_Slope = Up` , trafiamy do węzła, w którym (średnio) przeważa klasa „0” (brak choroby). Tam drzewo sprawdza kolejne zmienne:

  ChestPainType – czy jest jednym z typów `ATA, NAP, TA` ,

  Oldpeak (np. 0.45),

  Cholesterol 43.

  W ten sposób powstają kilka **liści** z ostatecznymi predykcjami – głównie klasa „0”, choć przy określonej kombinacji (`Oldpeak < 0.45` i `Cholesterol >= 43`) w jednym z węzłów dominuje już klasa „1”.

Jeżeli `ST_Slope` **nie jest „Up”** (czyli ma wartość „Flat” lub „Down”), przechodzimy do innej gałęzi, gdzie model również sprawdza `ChestPainType` ATA, NAP, TA oraz `MaxHR` 137. Tutaj w większości **liści** przewidywana jest klasa „1” (choroba), a jedynie przy pewnych wartościach `MaxHR` i `ChestPainType` drzewo zmienia predykcję na „0”.

W **liściach** (końcach gałęzi) widać przewidywaną klasę 0 lub 1, a także **procent obserwacji należących do klasy „1”** oraz **odsetek** wszystkich danych w tym segmencie. Dzięki temu łatwo sprawdzić, w których warunkach rośnie prawdopodobieństwo choroby (np. `ST_Slope != Up` i `ChestPainType ≠ ATA/NAP/TA` – aż 91% klasy „1”), a kiedy dominuje brak choroby (np. `ST_Slope = Up` i `ChestPainType ≠ ATA/NAP/TA` – tylko 7% klasy „1”). Drzewo pokazuje więc, jak krok po kroku zmienne takie jak **nachylenie ST, typ bólu w klatce, poziom cholesterolu czy maksymalne tętno** wpływają na końcową predykcję.

Finalne oceny

```
# Finalne oceny
final_predictions <- predict(best_model, newdata = testData, type = "class")
final_probabilities <- predict(best_model, newdata = testData, type = "prob")[, 2]

# Finalna macierz pomyłek
final_conf_matrix <- confusionMatrix(final_predictions, testData$HeartDisease)

# Obliczenie ROC i AUC
roc_curve <- roc(testData$HeartDisease, final_probabilities, levels = levels(testData$HeartDisease))
final_auc <- auc(roc_curve)

# Wyświetlanie wyników
print(final_conf_matrix)
cat("Final AUC: ", final_auc, "\n")
```

```
Confusion Matrix and Statistics

          Reference
Prediction   0   1
         0 109  24
         1  14 128

               Accuracy : 0.8618
                 95% CI : (0.8153, 0.9003)
    No Information Rate : 0.5527
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.7227

 Mcnemar's Test P-Value : 0.1443

            Sensitivity : 0.8862
            Specificity : 0.8421
         Pos Pred Value : 0.8195
         Neg Pred Value : 0.9014
             Prevalence : 0.4473
         Detection Rate : 0.3964
   Detection Prevalence : 0.4836
      Balanced Accuracy : 0.8641

       'Positive' Class : 0
Final AUC:  0.9162923
```

Parametry i wyniki modelu

- **Najlepsze parametry** Na podstawie najwyższej dokładności Accuracy) modelu, optymalne parametry wybrane z siatki poszukiwań to:
  - Maksymalna głębokość drzewa (maxdepth): 9
  - Minimalna liczba obserwacji do podziału w węźle (minsplit): 5
  - Parametr złożoności (cp): 0.011
- **Najlepsze metryki podczas walidacji krzyżowej CV**
  - **Accuracy (dokładność)** 82.26%
  - **AUC (pole pod krzywą ROC** 88.72%
  - **Precision (precyzja)** 80.50%
  - **Recall (czułość)** 80.20%

Wydajność końcowa na zbiorze testowym

Po zastosowaniu wytrenowanego modelu na danych testowych, uzyskano następujące wyniki:

Accuracy 86.18% (model poprawnie klasyfikuje ponad 86% obserwacji).

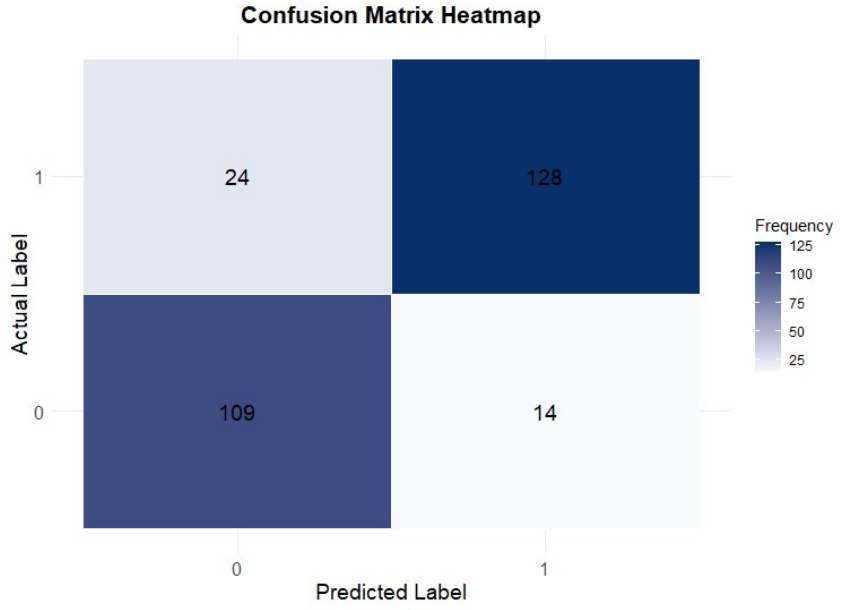
AUC 91.63% (model dobrze rozróżnia klasy w danych).

Kappa 0.7227 (model charakteryzuje się solidnym poziomem zgodności pomiędzy przewidywaniami a rzeczywistymi wartościami).

Sensivity (czułość) 88.62% (model dobrze identyfikuje obserwacje należące do klasy pozytywnej, tj. pacjentów z chorobą serca).

- **Specificity (specyficzność)**: 84.21% (model równie dobrze radzi sobie z identyfikacją pacjentów zdrowych).
- **Interpretacja wyników**
- Model drzewa decyzyjnego osiągnął bardzo dobre wyniki w klasyfikacji pacjentów pod kątem obecności choroby serca. Wysoka wartość AUC 91.63% wskazuje, że model skutecznie rozróżnia pacjentów z chorobą serca i bez niej.
- Osiągnięta precyzja 80.50% i czułość 88.62% wskazują na równowagę pomiędzy unikanie fałszywych alarmów (fałszywych pozytywów) a skutecznością w wykrywaniu faktycznych przypadków choroby.
- Konwersja macierzy pomyłek na tabelę i rysowanie heatmapy
- ```
  # Konwersja macierzy pomyłek na tabelę i rysowanie heatmapy
  conf_matrix <- as.table(final_conf_matrix$table)
  conf_matrix_df <- as.data.frame(conf_matrix)

  # Tworzenie heatmapy z liczbami
  ggplot(data = conf_matrix_df, aes(x = Prediction, y = Reference)) +
    geom_tile(aes(fill = Freq), color = "white") +
    scale_fill_gradient(low = "#f7fbff", high = "#08306b") +
    geom_text(aes(label = Freq), color = "black", size = 5) +
    labs(
      title = "Confusion Matrix Heatmap",
      x = "Predicted Label",
      y = "Actual Label",
      fill = "Frequency"
    ) +
    theme_minimal() +
    theme(
      plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
      axis.title = element_text(size = 14),
      axis.text = element_text(size = 12)
    )
  ```



Analiza algorytmu lasu losowego dla klasyfikacji

Tworzenie siatki parametrów

```
param_grid <- expand.grid(
  mtry = c(2, 4, 6),
  ntree = seq(100, 1000, by = 100),
  nodesize = c(1, 5, 10)
)
```
```

Funkcja do trenowania modelu i obliczania metryk z cross-walidacją

```
# Funkcja do trenowania modelu i obliczania metryk z cross-walidacją
evaluate_rf_model_cv <- function(params) {
  # Cross-walidacja
  folds <- createFolds(trainData$HeartDisease, k = 5, list = TRUE)

  metrics <- lapply(folds, function(fold_idx) {
    # Podział na dane treningowe i walidacyjne
    train_fold <- trainData[-fold_idx, ]
    val_fold <- trainData[fold_idx, ]

    # Trenowanie modelu
    model <- randomForest(
      HeartDisease ~ .,
      data = train_fold,
      mtry = as.numeric(params["mtry"]),
      ntree = as.numeric(params["ntree"]),
      nodesize = as.numeric(params["nodesize"])
    )

    # Predykcja na zbiorze walidacyjnym
    val_predictions <- predict(model, newdata = val_fold, type = "response")

    # Obliczanie metryk
    val_roc_curve <- roc(val_fold$HeartDisease, as.numeric(val_predictions))
```

```
    val_auc <- auc(val_roc_curve)
    cm <- confusionMatrix(val_predictions, val_fold$HeartDisease)
    precision <- cm$byClass["Precision"]
    recall <- cm$byClass["Recall"]
    accuracy <- mean(val_predictions == val_fold$HeartDisease)

    return(c(Accuracy = accuracy, AUC = val_auc, Precision = precision, Recall = recall))
  })

  # Średnie wyniki cross-walidacji
  mean_metrics <- colMeans(do.call(rbind, metrics))
  return(mean_metrics)
}
```

Cross-walidacja dla każdego zestawu hiperparametrów oraz wyświetlenie najlepszych parametrów

```
# Cross-walidacja dla każdego zestawu hiperparametrów
cv_metrics <- t(apply(param_grid, 1, function(row) evaluate_rf_model_cv(as.list(row))))

# Dodanie wyników do tabeli
results_rf <- cbind(param_grid, cv_metrics)

# Wyświetlenie najlepszych parametrów
best_rf_params <- results_rf[which.max(results_rf$Accuracy), ]
print("Najlepsze parametry:")
print(best_rf_params)
```

```
   mtry ntree nodesize  Accuracy       AUC Precision.Precision Recall.Recall
13    2   500        1 0.8740552 0.8710405           0.8711692       0.8432547
```

Trenowanie modelu z najlepszymi parametrami na całym zbiorze treningowym

```
# Trenowanie modelu z najlepszymi parametrami na całym zbiorze treningowym
final_rf_model <- randomForest(
  HeartDisease ~ .,
  data = trainData,
  mtry = best_rf_params$mtry,
  ntree = best_rf_params$ntree,
  nodesize = best_rf_params$nodesize
)
```

Predykcja , obliczenie metryk i wyświetlenie wyników na zbiorze testowym

```
# Predykcja na zbiorze testowym
final_test_predictions <- predict(final_rf_model, newdata = testData, type = "response")

# Obliczenie metryk na zbiorze testowym
final_roc_curve <- roc(testData$HeartDisease, as.numeric(final_test_predictions))
final_auc <- auc(final_roc_curve)
final_cm <- confusionMatrix(final_test_predictions, testData$HeartDisease)
final_precision <- final_cm$byClass["Precision"]
final_recall <- final_cm$byClass["Recall"]
final_accuracy <- mean(final_test_predictions == testData$HeartDisease)

# Wyświetlenie wyników na zbiorze testowym
cat("Metryki na zbiorze testowym:\n")
cat("Accuracy:", final_accuracy, "\n")
cat("AUC:", final_auc, "\n")
cat("Precision:", final_precision, "\n")
cat("Recall:", final_recall, "\n")
```

```
Metryki na zbiorze testowym:
Accuracy: 0.8836364
AUC: 0.8838789
Precision: 0.8582677
Recall: 0.8861789
```

1. Wybór parametrów

- **Najlepsze parametry**:
  - **mtry:** 2
  - **ntree:** 500
  - **nodesize:** 1

Te ustawienia zostały wybrane na podstawie najwyższej dokładności Accuracy), wynoszącej **87.40%**, przy AUC **87.10%** w crosswalidacji.

---

2. Ocena na zbiorze testowym

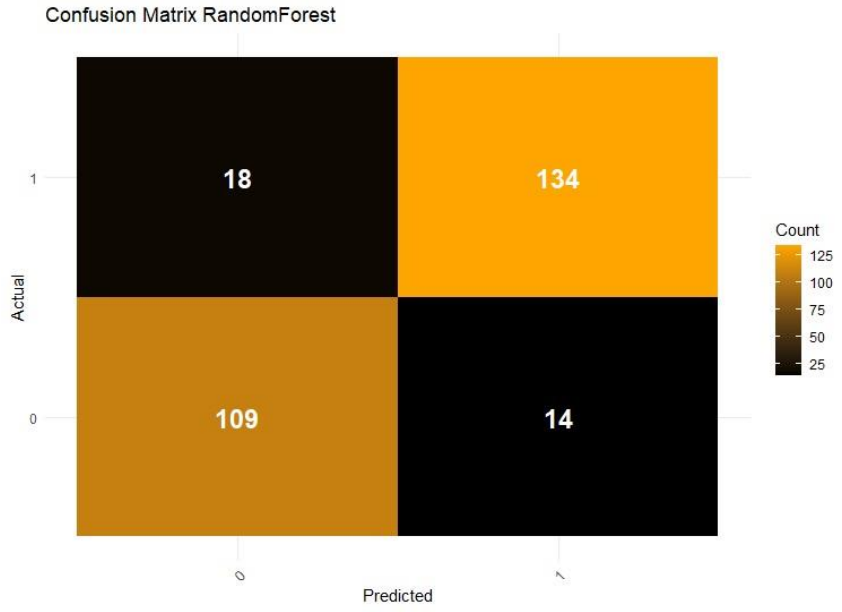
Po zastosowaniu wytrenowanego modelu na danych testowych, las losowy osiągnął następujące wyniki:

- **Accuracy (dokładność):** 88.36% (model poprawnie klasyfikuje ponad 88% obserwacji).
- **AUC (pole pod krzywą ROC** 88.38% (model efektywnie rozróżnia klasy w danych).
- **Precision (precyzja):** 85.83% (model unika fałszywych alarmów, czyli błędnych klasyfikacji pozytywnych).
- **Recall (czułość):** 88.62% (model skutecznie wykrywa obserwacje należące do klasy pozytywnej, np. pacjentów z chorobą serca).

Wyświetlenie confusion matrix jako heatmapy

```
# Wyświetlenie confusion matrix jako heatmapy
test_cm_matrix <- as.table(final_cm$table)
cm_data <- as.data.frame(test_cm_matrix)
colnames(cm_data) <- c("Predicted", "Actual", "Count")

# Tworzenie heatmapy z liczbami
ggplot(data = cm_data, aes(x = Predicted, y = Actual, fill = Count)) +
  geom_tile() +
  geom_text(aes(label = Count), color = "white", fontface = "bold", size = 6) +
  scale_fill_gradient(low = "black", high = "orange") +
  theme_minimal() +
  labs(title = "Confusion Matrix RandomForest", x = "Predicted", y = "Actual") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



Analiza algorytmu drzewa regresyjnego

Informacje datasecie “financial_regression.csv” zaczerpniętego z kaggle
Zestaw danych szeregów czasowych z informacjami finansowymi dla następujących elementów:

- S&P 500 – SPDR S&P 500 ETF Trust
- Nasdaq 100 – Invesco QQQ ETF
- Stopy procentowe w USA – miesięczne stopy federalne
- CPI – miesięczny wskaźnik cen towarów i usług konsumpcyjnych
- Kurs forex USD / CHF
- Kurs forex EUR/USD
- PKB – Produkt Krajowy Brutto, co trzy miesiące
- Srebro - abrdn Physical Silver Shares ETF
- Ropa naftowa – USO ETF
- Platyna - abrdn Physical Platinum Shares ETF
- Pallad - abrdn Physical Palladium Shares ETF
- Złoto – SPDR Gold Trust ETF

Wczytywanie danych

```
data <- read.csv("financial_regression.csv")
```

Usuwamy zmienne: date, gold.open, gold.high, gold.low, gold.volume

```
# Usuwamy zmienne: date, gold.open, gold.high, gold.low, gold.volume
data <- data[, !(names(data) %in% c("date", "gold.open", "gold.high", "gold.low", "gold.volume"))]
data <- data[!is.na(data$gold.close), ]
```

Sprawdzamy braki dla “gold.close”

```
sum(is.na(data["gold.close"]))
[1] 0
```

Dzielimy dane na zbiór treningowy i testowy

```
set.seed(123)
train_index <- sample(1:nrow(data), size = 0.7 * nrow(data))
train_data <- data[train_index, ]
test_data <- data[-train_index, ]
```

Funkcje imputujące brakujące wartości

```
# Funkcja imputująca brakujące wartości
train_data_imputed <- train_data %>%
  mutate(across(everything(), ~replace(., is.na(.), mean(., na.rm = TRUE))))

test_data_imputed <- test_data %>%
  mutate(across(everything(), ~replace(., is.na(.), mean(., na.rm = TRUE))))
```

Funkcja do trenowania modelu i obliczania metryk z cross-walidacją

```
evaluate_model_cv_rpart <- function(params, trainData, folds = 10) {
  # Przygotowanie zbiorów do cross-walidacji
  cv_folds <- createFolds(trainData$gold.close, k = folds, list = TRUE)

  # Miejsce na metryki
  metrics <- data.frame(RMSE = numeric(), MAE = numeric(), R2 = numeric(), MAPE = numeric())

  # Cross-walidacja
  for (fold in cv_folds) {
    train_fold <- trainData[-fold, ]
    test_fold <- trainData[fold, ]

    # Trenowanie modelu (drzewo decyzyjne)
    model <- rpart(
      gold.close ~ .,
      data = train_fold,
      method = "anova",  # Regresja
      control = rpart.control(
        cp = as.numeric(params["cp"]),
        maxdepth = as.numeric(params["maxdepth"]),
        minsplit = as.numeric(params["minsplit"])
      )
    )

    # Predykcja na zbiorze testowym
    predictions <- predict(model, newdata = test_fold)

    # Obliczanie metryk
    rmse <- sqrt(mean((predictions - test_fold$gold.close)^2))
    mae <- mean(abs(predictions - test_fold$gold.close))
    r2 <- cor(predictions, test_fold$gold.close)^2
    mape <- mean(abs((predictions - test_fold$gold.close) / test_fold$gold.close)) * 100

    # Zapis metryk
    metrics <- rbind(metrics, data.frame(RMSE = rmse, MAE = mae, R2 = r2, MAPE = mape))
  }

  # Średnie metryki z CV
  return(colMeans(metrics, na.rm = TRUE))
}
```

Parametry do grid search dla drzewa decyzyjnego oraz zmienna, która będzie przechowywać wyniki

```
# Parametry do grid searcha dla drzewa decyzyjnego
grid_search_rpart <- expand.grid(
  cp = seq(0.01, 0.1, by = 0.01),
  maxdepth = c(5, 10, 15),
  minsplit = c(10, 20, 30)
)

# Zmienna, która będzie przechowywać wyniki
best_rpart_metrics <- data.frame()
```

Grid search, iteracja po parametrach

```
# Grid search, iteracja po parametrach
for (i in 1:nrow(grid_search_rpart)) {
  params <- grid_search_rpart[i, ]
  metrics <- evaluate_model_cv_rpart(params, train_data_imputed, folds = 10)

  # Połączenie parametrów z metrykami bez użycia cbind
  result <- data.frame(
    cp = params$cp,
    maxdepth = params$maxdepth,
    minsplit = params$minsplit,
    RMSE = metrics["RMSE"],
    MAE = metrics["MAE"],
    R2 = metrics["R2"],
    MAPE = metrics["MAPE"]
  )

  # Dodawanie wyniku do ramki danych (usuńmy nazwy wierszy)
  best_rpart_metrics <- bind_rows(best_rpart_metrics, result)
}
```

Wyświetlenie najlepszych parametrów dla drzewa decyzyjnego (minimalny RMSE

```
# Wyświetlenie najlepszych parametrów dla drzewa decyzyjnego (minimalny RMSE)
best_rpart_metrics_clean <- best_rpart_metrics
rownames(best_rpart_metrics_clean) <- NULL
best_rpart_metrics_clean[which.min(best_rpart_metrics_clean$RMSE), ]

     cp maxdepth minsplit     RMSE      MAE        R2     MAPE
11 0.01       10       10 9.139367 6.958001 0.9047769 4.769911
```

Trenowanie modelu z najlepszymi parametrami i obliczanie metryk

```
# Obliczanie metryk na zbiorze testowym
final_rmse <- sqrt(mean((final_predictions - test_data_imputed$gold.close)^2))
final_mae <- mean(abs(final_predictions - test_data_imputed$gold.close))
final_r2 <- cor(final_predictions, test_data_imputed$gold.close)^2
final_mape <- mean(abs((final_predictions - test_data_imputed$gold.close) / test_data_imputed$gold.cl
ose)) * 100

cat("Test Set Metrics:\n")
cat("RMSE:", final_rmse, "\n")
cat("MAE:", final_mae, "\n")
cat("R2:", final_r2, "\n")
cat("MAPE:", final_mape, "%\n")
Test Set Metrics:
```

```
> cat("RMSE:", final_rmse, "\n" )
RMSE: 9.136226
> cat("MAE:", final_mae, "\n" )
MAE: 6.871348
> cat("R2:", final_r2, "\n" )
R2: 0.9048563
> cat("MAPE:", final_mape, "%\n" )
MAPE: 4.691194 %
```

1. Wybór parametrów

- **Najlepsze parametry drzewa regresyjnego** (na podstawie minimalnego RMSE w cross-walidacji):
  - **cp (parametr złożoności):** 0.01
  - **maxdepth (maksymalna głębokość drzewa):** 10
  - **minsplit (minimalna liczba obserwacji w węźle):** 10

Te parametry zapewniły najlepsze wyniki w walidacji krzyżowej:

- **RMSE** 9.139367
- **MAE** 6.958001
- **R^2:** 90.48%
- **MAPE** 4.77%

---

2. Wydajność na zbiorze testowym

Po zastosowaniu modelu drzewa regresyjnego z najlepszymi parametrami na zbiorze testowym uzyskano następujące wyniki:

- **RMSE (średni błąd kwadratowy):**9.136226
- **MAE (średni błąd absolutny):**6.871348
- **R^2 (współczynnik determinacji):** 90.49%
- **MAPE (średni procentowy błąd bezwzględny):**4.69%

---

3. Interpretacja wyników

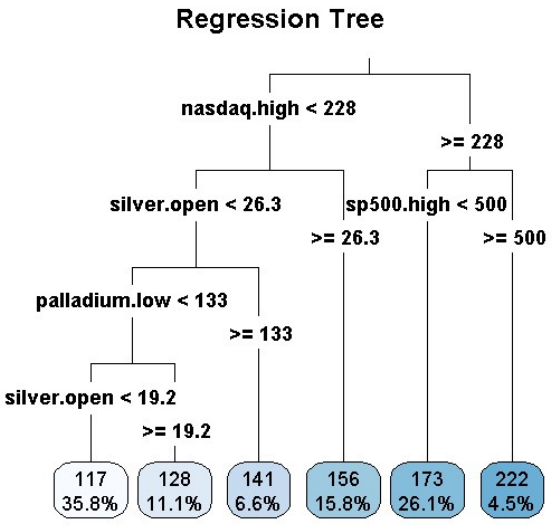
Model drzewa regresyjnego osiągnął bardzo dobre wyniki w przewidywaniu zmiennej docelowej:

- **R^2 wynoszące 90.49%** oznacza, że model wyjaśnia ponad 90% zmienności zmiennej zależnej, co wskazuje na wysoką jakość dopasowania.
- **Niskie RMSE 9.14** i **MAE 6.87** świadczą o tym, że przewidywane wartości są bliskie rzeczywistym wartościom w testowym zestawie danych.
- **MAPE na poziomie 4.69%** potwierdza precyzję modelu, ponieważ średni błąd względny w prognozach wynosi mniej niż 5%.
-

Wizualizacja wyników

```
# Wizualizacja wyników
results_df <- data.frame(
  Actual = test_data_imputed$gold.close,
  Predicted = final_predictions
)

rpart.plot(
  best_model,
  type = 3,
  digits = 3,
  fallen.leaves = TRUE,
  main = "Regression Tree"
)
```



„Powyższe drzewo regresyjne rozpoczyna podział danych od warunku na zmiennej `nasdaq.high` . Jeżeli wartość `nasdaq.high` jest niższa niż 228, w kolejnym kroku model dzieli obserwacje według `silver.open` . Następnie,

w zależności od wartości `palladium.low` i ponownie `silver.open` , uzyskujemy trzy finalne liście z odpowiednimi predykcjami (średnimi wartościami zmiennej docelowej). Dla obserwacji z `nasdaq.high >= 228` drzewo dzieli się z kolei według `sp500.high` , wyróżniając ostatecznie dwa liście (z prognozami ok. 173 i 222). Każdy liść pokazuje szacowaną średnią zmiennej objaśnianej w danej grupie oraz procent obserwacji, które do tej grupy należą.”

Wybór parametrów:

- Najlepsze parametry dla drzewa regresyjnego:
  - cp: 0.01 maxdepth:
  - 10 minsplit: 10
  - Te parametry osiągnęły minimalne **RMSE** wynoszące **9.14**, przy **MAE** na poziomie **6.96**, **R^2** 90.48% oraz **MAPE** 4.77%.
-

Ocena modelu:

- Drzewo regresyjne dobrze modeluje dane, osiągając wysokie R^2, co oznacza, że model wyjaśnia ponad 90% wariancji zmiennej zależnej.

Wady:

- Pomimo wysokiej interpretowalności, drzewo regresyjne może być mniej precyzyjne dla bardziej złożonych zależności w danych.

Analiza algorytmu lasu losowego dla regresji

Funkcja do trenowania modelu i obliczania metryk z cross-walidacją

```
evaluate_model_cv_rf <- function(params, trainData, folds = 10) {
  # Przygotowanie zbiorów do cross-walidacji
  cv_folds <- createFolds(trainData$gold.close, k = folds, list = TRUE)

  # Miejsce na metryki
  metrics <- data.frame(RMSE = numeric(), MAE = numeric(), R2 = numeric(), MAPE = numeric())

  # Cross-walidacja
  for (fold in cv_folds) {
    train_fold <- trainData[-fold, ]
    test_fold <- trainData[fold, ]

    # Trenowanie modelu (las losowy)
    model <- randomForest(
      gold.close ~ .,
      data = train_fold,
      mtry = as.numeric(params["mtry"]),
      ntree = as.numeric(params["ntree"]),
      nodesize = as.numeric(params["nodesize"])
    )

    # Predykcja na zbiorze testowym
    predictions <- predict(model, newdata = test_fold)

    # Obliczanie metryk
    rmse <- sqrt(mean((predictions - test_fold$gold.close)^2))
    mae <- mean(abs(predictions - test_fold$gold.close))
    r2 <- cor(predictions, test_fold$gold.close)^2
    mape <- mean(abs((predictions - test_fold$gold.close) / test_fold$gold.close)) * 100

    # Zapis metryk
    metrics <- rbind(metrics, data.frame(RMSE = rmse, MAE = mae, R2 = r2, MAPE = mape))
  }

  # Średnie metryki z CV
  return(colMeans(metrics, na.rm = TRUE))
}
```

Parametry do grid searcha dla Random Forest

```
# Parametry do grid searcha dla Random Forest
grid_search_rf <- expand.grid(
  mtry = c(2, 3, 4),
  ntree = c(50, 100, 150),
  nodesize = c(5, 10, 20)
```

Grid search, iteracja po parametrach

```
# Zmienna, która będzie przechowywać wyniki
best_rf_metrics <- data.frame()

# Grid search, iteracja po parametrach
for (i in 1:nrow(grid_search_rf)) {
  params <- grid_search_rf[i, ]
  metrics <- evaluate_model_cv_rf(params, train_data_imputed, folds = 10)

  # Połączenie parametrów z metrykami bez użycia cbind
  result <- data.frame(
    mtry = params$mtry,
    ntree = params$ntree,
    nodesize = params$nodesize,
    RMSE = metrics["RMSE"],
    MAE = metrics["MAE"],
    R2 = metrics["R2"],
    MAPE = metrics["MAPE"]
  )

  # Dodawanie wyniku do ramki danych
  best_rf_metrics <- bind_rows(best_rf_metrics, result)
}
```

Wyświetlenie najlepszych parametrów dla Random Forest (minimalny RMSE

```
# Wyświetlenie najlepszych parametrów dla Random Forest (minimalny RMSE)
best_rf_metrics_clean <- best_rf_metrics
rownames(best_rf_metrics_clean) <- NULL
best_rf_metrics_clean[which.min(best_rf_metrics_clean$RMSE), ]

  mtry ntree nodesize    RMSE      MAE        R2     MAPE
6    4   100         5 2.19541 1.502465 0.9946439 1.015975
```

```
# Trenowanie modelu z najlepszymi parametrami
best_rf_model <- randomForest(
  gold.close ~ .,
  data = train_data_imputed,
  mtry = best_rf_params$mtry,
  ntree = best_rf_params$ntree,
  nodesize = best_rf_params$nodesize
)

# Predykcja na zbiorze testowym
rf_predictions <- predict(best_rf_model, newdata = test_data_imputed)

# Ewaluacja modelu na zbiorze testowym
rf_rmse <- sqrt(mean((rf_predictions - test_data_imputed$gold.close)^2))
rf_mae <- mean(abs(rf_predictions - test_data_imputed$gold.close))
rf_r2 <- cor(rf_predictions, test_data_imputed$gold.close)^2
rf_mape <- mean(abs((rf_predictions - test_data_imputed$gold.close) / test_data_imputed$gold.close))
* 100

# Wyświetlenie metryk
cat("Random Forest Test Set Evaluation:\n")
cat(sprintf("RMSE: %.3f\n", rf_rmse))
cat(sprintf("MAE: %.3f\n", rf_mae))
cat(sprintf("R^2: %.3f\n", rf_r2))
cat(sprintf("MAPE: %.3f%%\n", rf_mape))
> cat("Random Forest Test Set Evaluation:\n" )
Random Forest Test Set Evaluation:
> cat(sprintf("RMSE: %.3f\n", rf_rmse ))
RMSE: 2.037
> cat(sprintf("MAE: %.3f\n", rf_mae ))
MAE: 1.410
> cat(sprintf("R^2: %.3f\n", rf_r 2))
R^2: 0.995
> cat(sprintf("MAPE: %.3f%%\n", rf_mape ))
MAPE: 0.938%
```

1. Wybór parametrów

- **Najlepsze parametry lasu losowego**(na podstawie minimalnego RMSE w cross-walidacji):
  - **mtry (liczba zmiennych branych pod uwagę na węzeł):**4
  - **ntree (liczba drzew w lesie):** 100
  - **nodesize (minimalna liczba obserwacji w węźle):**5

Te parametry zapewniły najlepsze wyniki w walidacji krzyżowej:

- **RMSE** 2.19541
- **MAE** 1.502465
- **R²:** 99.46%
- **MAPE** 1.02%

---

2. Wydajność na zbiorze testowym

Po zastosowaniu modelu lasu losowego na zbiorze testowym osiągnięto następujące wyniki:

- **RMSE (średni błąd kwadratowy):**2.037
- **MAE (średni błąd absolutny):** 1.410
- **R² (współczynnik determinacji):** 99.50%
- **MAPE (średni procentowy błąd bezwzględny):**0.94%

---

3. Interpretacja wyników

Model lasu losowego wykazał doskonałą zdolność przewidywania zmiennej docelowej:

- **R² na poziomie 99.50%** wskazuje, że model niemal całkowicie wyjaśnia zmienność zmiennej zależnej, co świadczy o bardzo wysokim dopasowaniu.
- **Niskie RMSE 2.037** i **MAE 1.410** potwierdzają, że prognozowane wartości są niezwykle bliskie rzeczywistym wartościom w zbiorze testowym.

  MAPE wynoszące 0.94% sugeruje, że model popełnia minimalne błędy w przewidywaniach, co czyni go wysoce precyzyjnym.

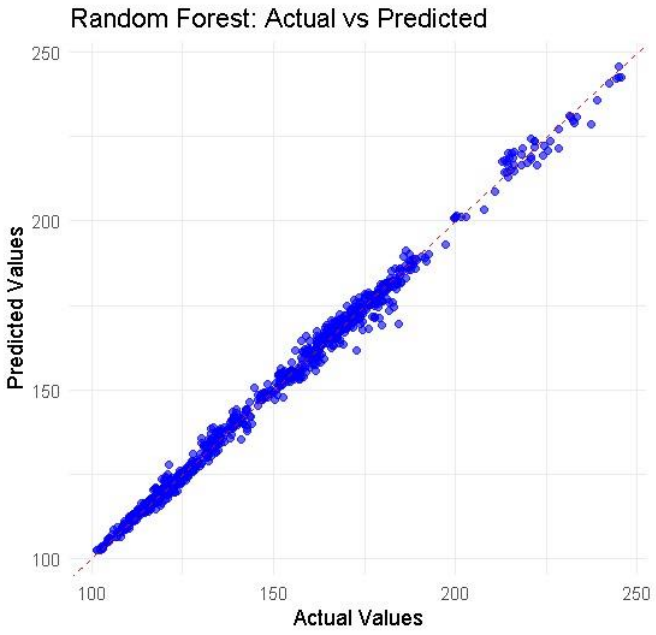
4. Porównanie z drzewem regresyjnym

W porównaniu z modelem drzewa regresyjnego, las losowy osiągnął znacznie lepsze wyniki:

- **RMSE**: Las losowy (2.037) jest znacznie mniejszy niż w drzewie regresyjnym (9.136226), co oznacza wyższą precyzję prognoz.
- **R^2:** Las losowy (99.50%) znacznie przewyższa drzewo regresyjne (90.49%), co wskazuje na lepsze wyjaśnienie zmienności danych.

  MAPE: Las losowy (0.94%) jest bardziej precyzyjny niż drzewo regresyjne (4.69%).
-

Wykres Rozrzutu (wartości rzeczywiste vs przewidywane):



Podsumowanie

W analizie wykorzystano drzewa decyzyjne i lasy losowe do klasyfikacji i regresji. Dla klasyfikacji (wykrywanie chorób serca) las losowy osiągnął lepsze wyniki niż drzewo decyzyjne, z wyższą dokładnością (**88.36% vs 86.18%**) i większą odpornością na przeuczenie.

W zadaniach regresyjnych (prognozowanie cen złota) las losowy zdecydowanie przewyższył drzewo regresyjne, osiągając znacznie mniejsze RMSE (**2.037 vs 9.136**) oraz wyższe R^2 (**99.50% vs 90.49%**).

Drzewa decyzyjne są bardziej interpretowalne, ale w zadaniach wymagających wysokiej precyzji i stabilności las losowy okazał się lepszym wyborem.