# Projekt zaliczeniowy

# Techniki eksploracji danych

Paulina Górska
Informatyka i Ekonometria
Nr albumu: 185203

# Zakres projektu

Założeniem projektu było rozwiązanie trzech problemów:

1) klasyfikacji binarnej
2) klasyfikacji wieloklasowej
3) regresji.

Rozwiązania powyższych problemów należało dokonać za pomocą własnych implementacji algorytmów:

- k-najbliższych sąsiadów
- drzewa decyzyjne
- maszyna wektorów nośnych (tylko dla problemu klasyfikacji binarnej)
- sieci neuronowe.

Otrzymane wyniki dla własnych implementacji algorytmów należało porównać z wynikami otrzymanymi dla algorytmów opracowanych na podstawie wbudowanych pakietów języka R.

Analiza powinna obejmować wpływ hiper-parametrów na jakość opracowanych modeli oraz porównanie wyników najlepszych modeli , własnych jak i tych wbudowanych.

# Zbiory danych

1) Klasyfikacja binarna:
   https://archive.ics.uci.edu/ml/datasets/Caesarian+Section+Classification+Dataset
   - Liczba obserwacji: 80
   - Liczba atrybutów: 5
   - Zmienna celu: zmienna binarna (0 – nie było cesarskiego cięcia(nie); 1 – było cesarskie cięcie (tak))

2) Klasyfikacja wieloklasowa:
   https://archive.ics.uci.edu/ml/datasets/Balance+Scale
   - Liczba obserwacji: 625
   - Liczba atrybutów: 4
   - Zmienna celu: trzy klasy (L, B, R)

3) Regresja:
   https://archive.ics.uci.edu/ml/datasets/Servo
   - Liczba obserwacji: 167
   - Liczba atrybutów: 4
   - Zmienna celu: zmienna numeryczna

Wszystkie zbiory danych nie zawierały braków danych.

Przekształcenia danych:

- zmienna celu w problemie klasyfikacji wieloklasowej została przekształcona z oznaczeń literowych na wartości liczbowe: 1-B; 2-L; 3-R
- w problemie regresji dwie zmienne: „motor” oraz „screw” zostały przekształcone na wartości liczbowe (1,2,3,4,5)
- dokonano normalizacji danych dla trzech algorytmów: k-najbliższych sąsiadów, maszyna wektorów nośnych oraz sieci neuronowe
- dla algorytmu maszyny wektorów nośnych przekształcono zmienną celu do wartości {-1;1}

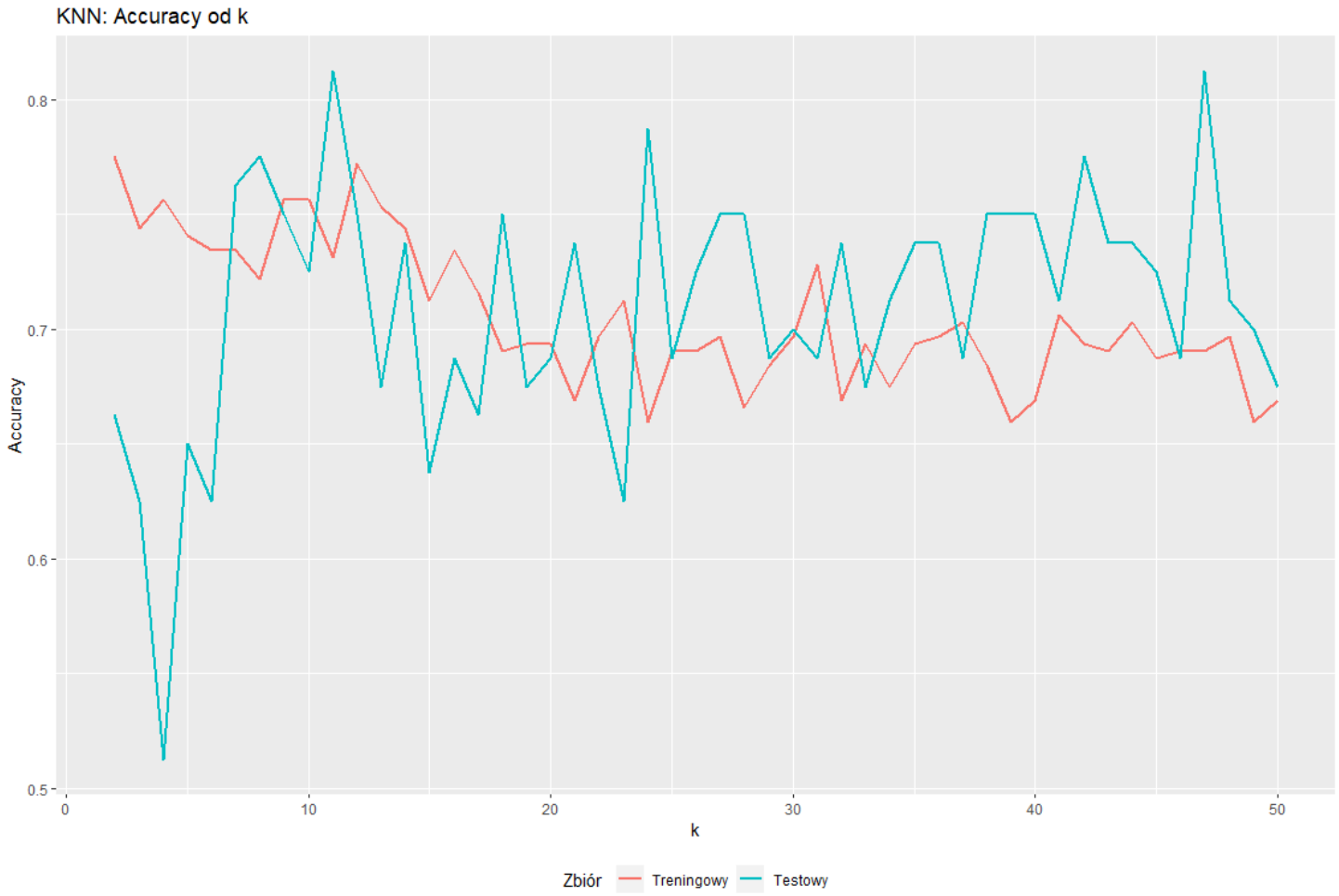
Kroswalidacja – W praktycznie wszystkich przypadkach użyto podziału danych z parametrem *kFold = 5*, ponieważ zbiory nie są duże.

## Najlepsze modele dla poszczególnych algorytmów

| | Algorytm | Implementacja | Parametry | Ocena jakości dla zbioru Testowego (Trafność) |
|---|---|---|---|---|
| **Klasyfikacja Binarna** | **knn** | **Własna** | **k = 11** | **0.8125** |
| | SVM | Własna | C = 30 | 0.7625 |
| | Sieci NN | Własna | h = (4,4), *iter* = 10000 | 0.6625 |
| | knn | Biblioteka R | k = 11 | 0.7321 |
| | SVM | Biblioteka R | C = 7 | 0.6519 |
| | Sieci NN | Biblioteka R | h = 5 | 0.6149 |
| | Drzewa decyzyjne | Biblioteka R | max depth = 3 | 0.6545 |
| **Klasyfikacja Wieloklasowa** | knn | Własna | k = 42 | 0.9194 |
| | Sieci NN | Własna | h = (6,6), *iter* = 80000 | 0.4736 |
| | knn | Biblioteka R | k = 9 | 0.9089 |
| | **Sieci NN** | **Biblioteka R** | **h = 9** | **0.9679** |
| | Drzewa decyzyjne | Biblioteka R | max depth = 12 | 0.7903 |
| | | | | **( MAE )** |
| **Regresja** | **knn** | **Własna** | **k = 2** | **0.2483** |
| | Sieci NN | Własna | h = (8,8), *iter* = 100000 | 0.4776 |
| | knn | Biblioteka R | k = 2 | 0.2658 |
| | Sieci NN | Biblioteka R | h = 6 | 0.9705 |
| | Drzewa decyzyjne | Biblioteka R | max depth = 3 | 0.4630 |

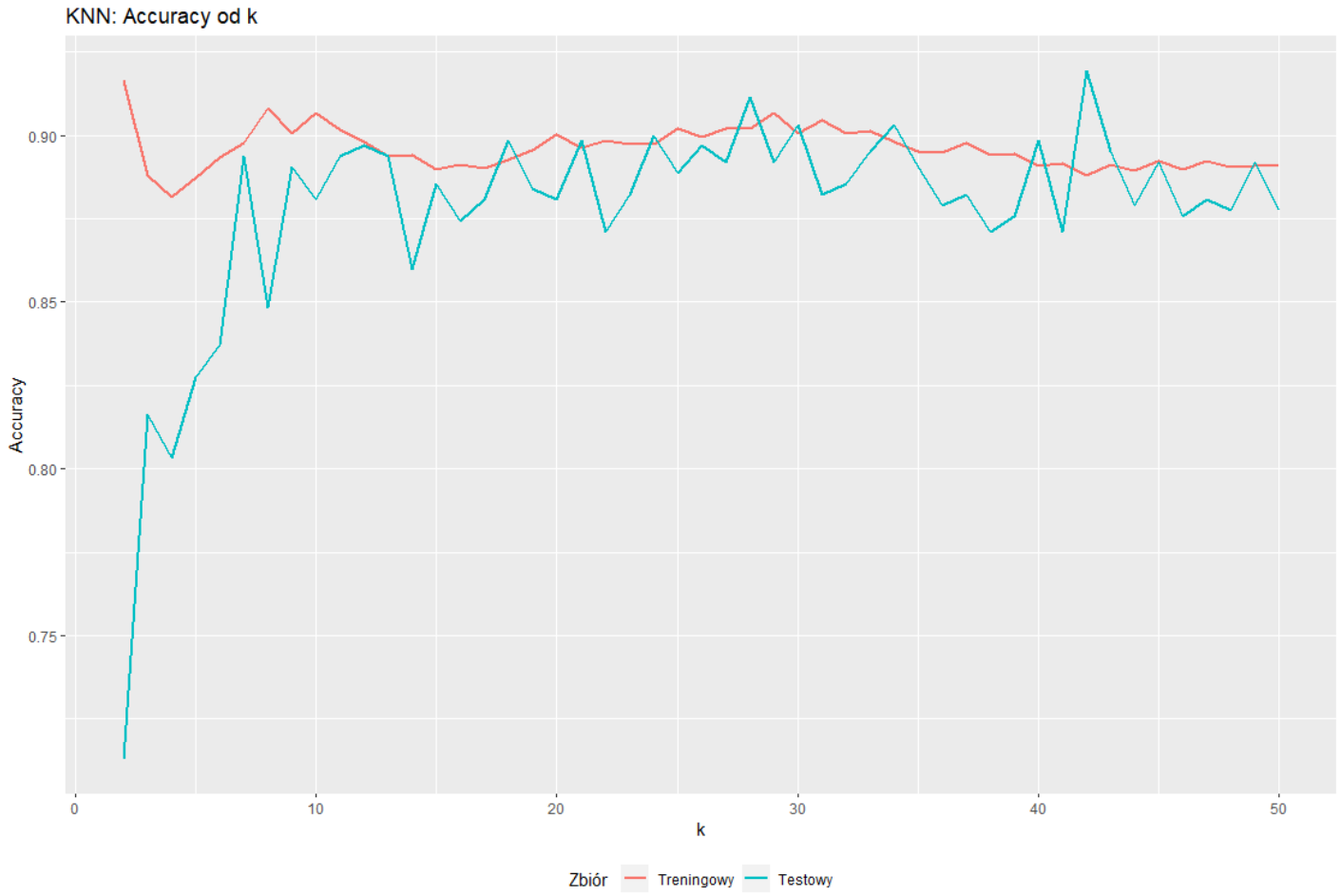
# k najbliższych sąsiadów - knn (własna implementacja)

## Klasyfikacja Binarna



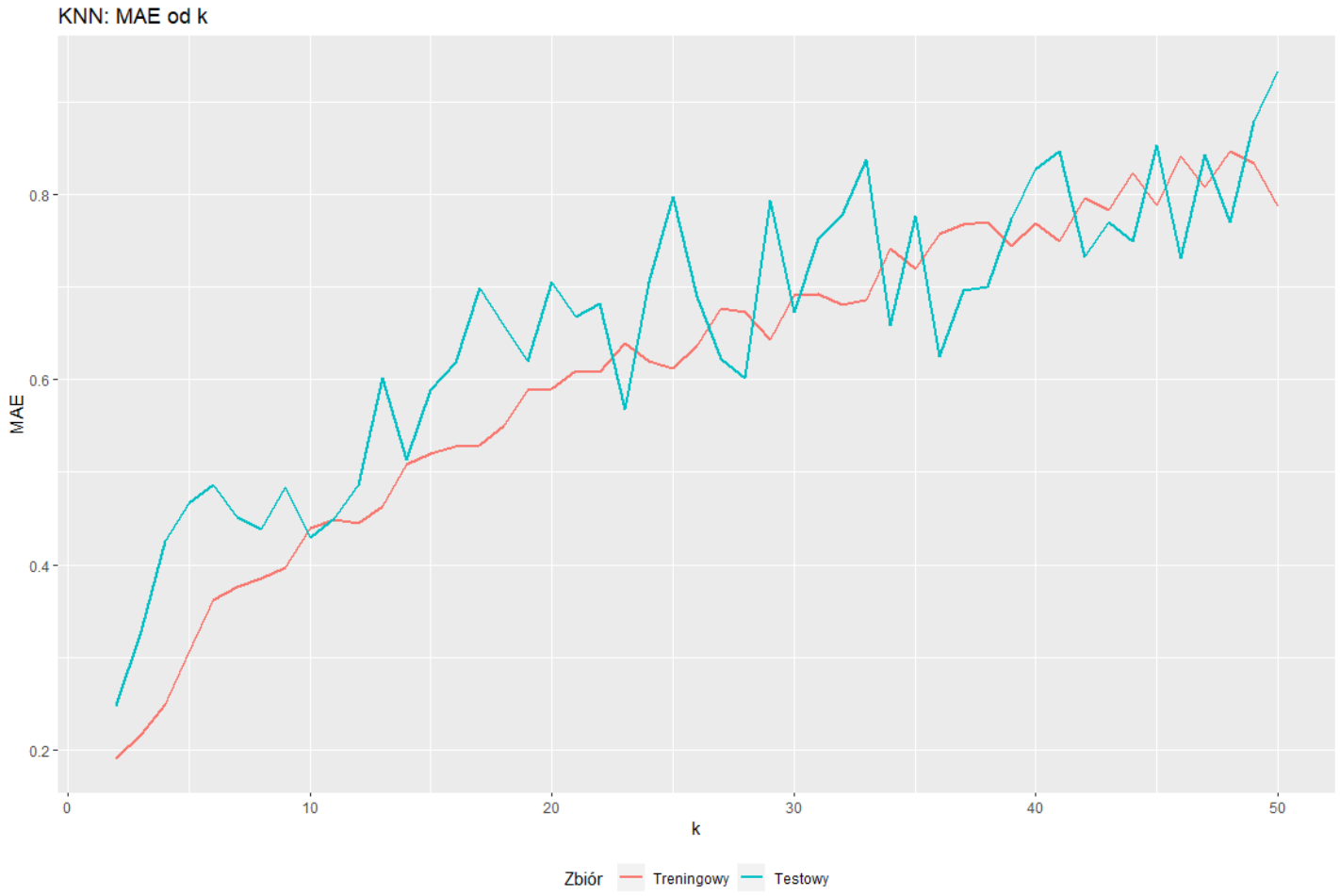
# k najbliższych sąsiadów - knn (własna implementacja)

## Klasyfikacja Wieloklasowa



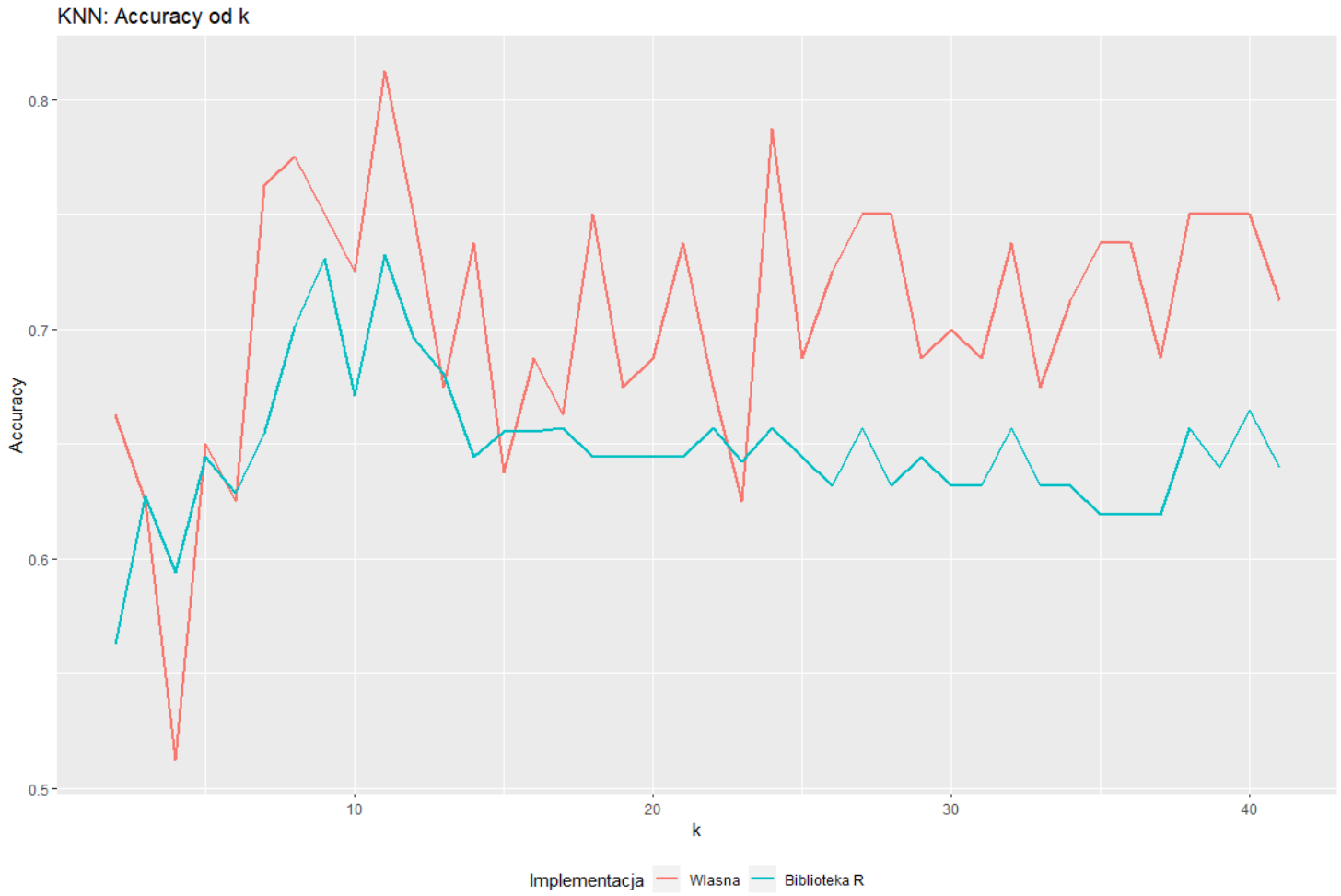
# k najbliższych sąsiadów – knn (własna implementacja)

## Regresja



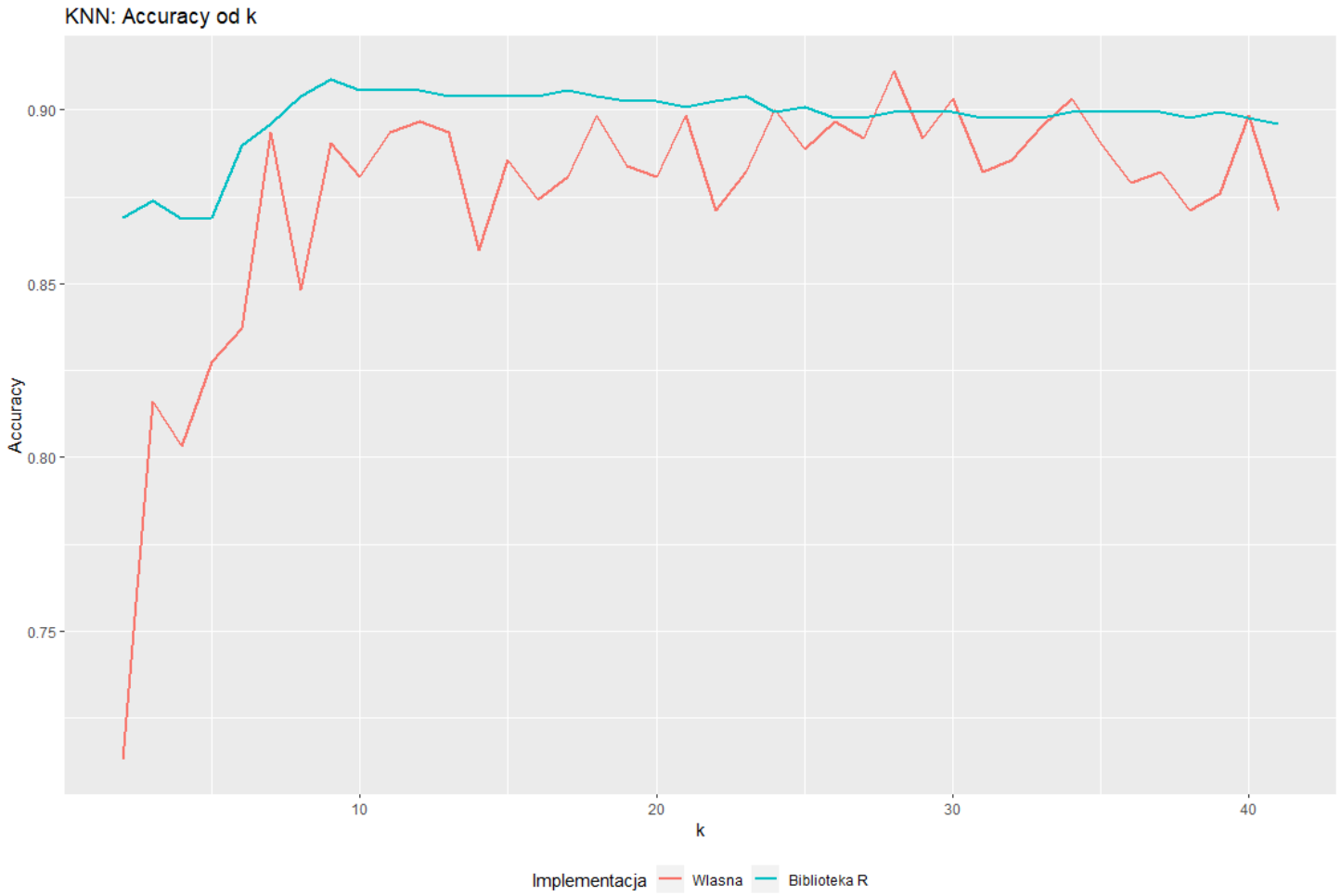
# k-nn– Porównanie implementacji

## Klasyfikacja Binarna



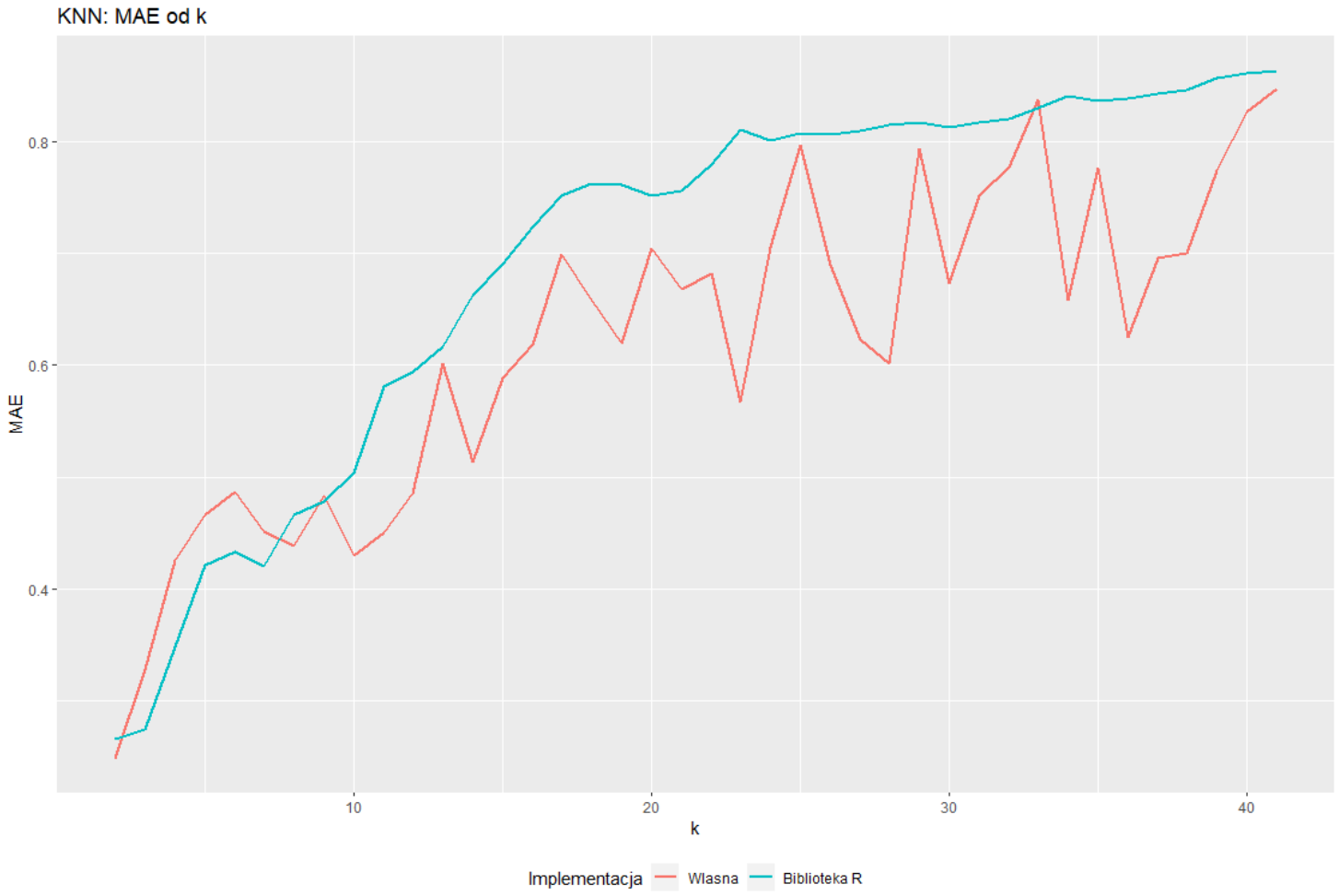
# k-nn – Porównanie implementacji

## Klasyfikacja Wieloklasowa



# k-nn – Porównanie implementacji

## Regresja



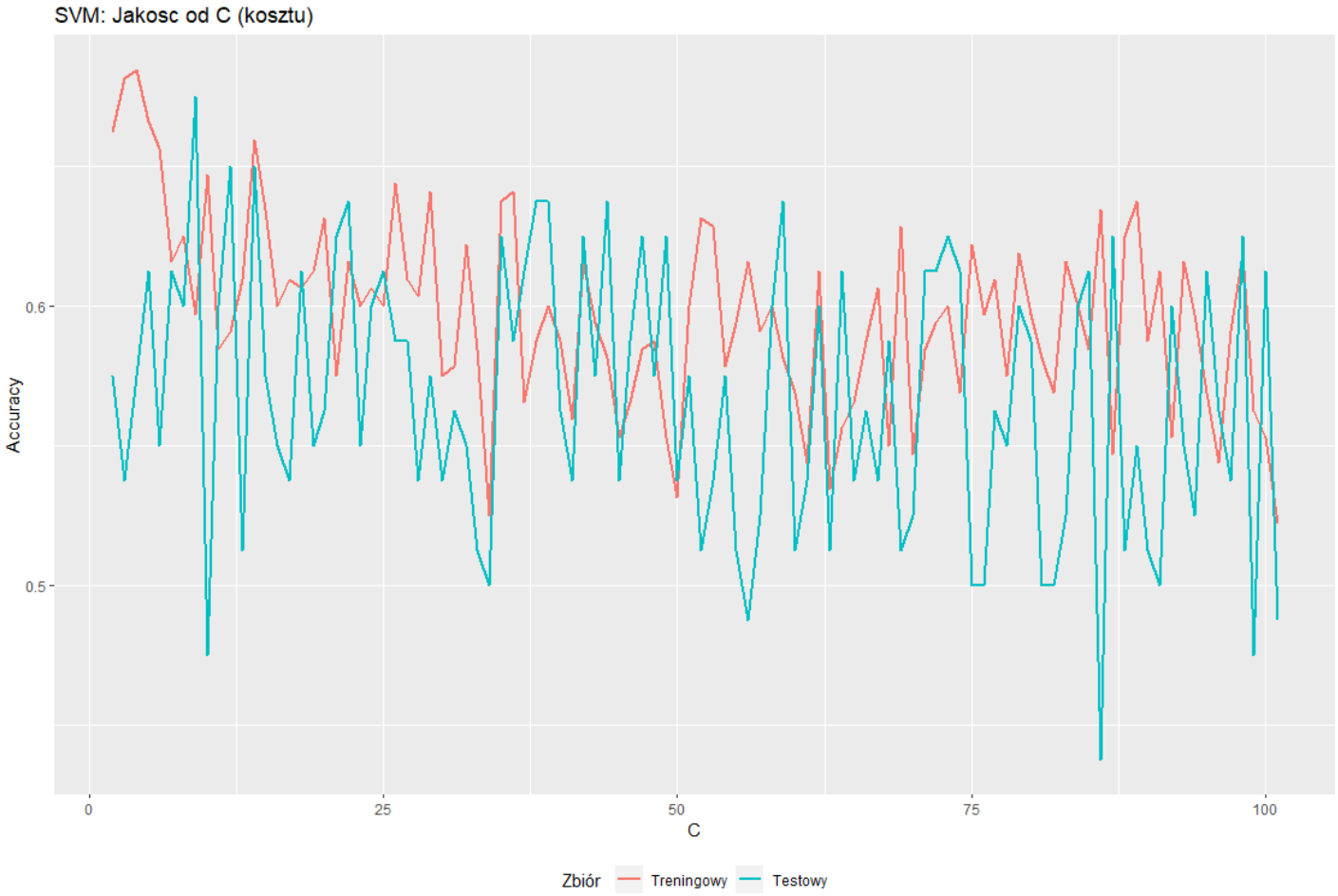
## Podsumowanie algorytmu k-nn:

Własna implementacja algorytmu k-nn okazała się, nie być najgorszą implementacją tego algorytmu – w przypadku klasyfikacji binarnej, ideowy algorytm, lepiej poradził sobie z klasyfikacją niż zaimplementowany w bibliotece dla R. W pozostałych przypadkach to jednak algorytmy z biblioteki były szybsze i nie powodowały dużych oscylacji Trafności czy MAE dla kolejnych wartości sąsiadów k.
Szybkość algorytmów nie była tutaj brana pod uwagę, ale podczas testów dało się odczuć w tym aspekcie przewagę implementacji algorytmu z biblioteki.

Algorytm k-nn można usprawnić na kilka sposobów – w zależności od implementacji, klasyfikator na bazie k-nn, będzie uzyskiwał różne czasy wykonania oraz różną skuteczność.

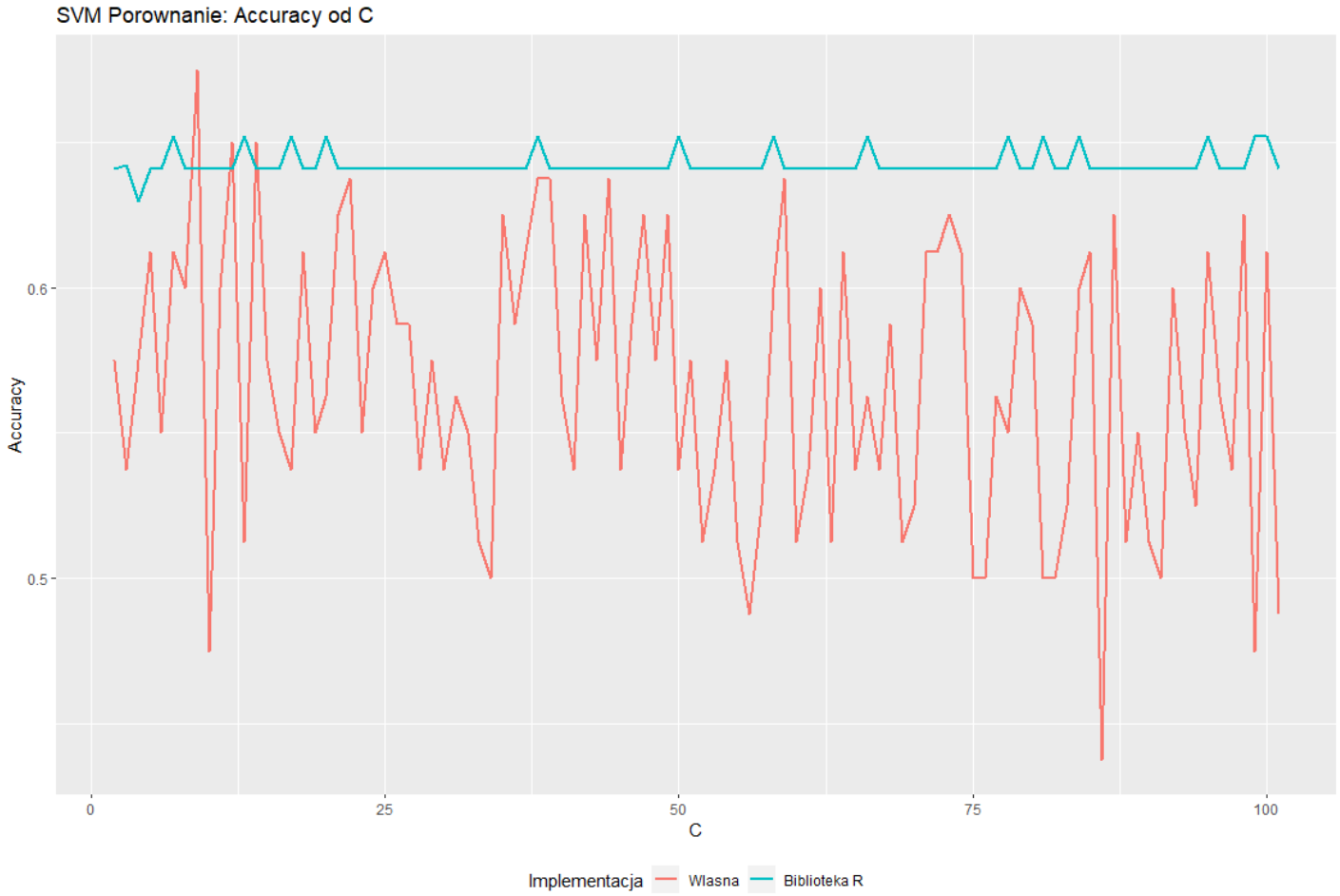
# SVM (własna implementacja)

## Klasyfikacja Binarna



# SVM – Porównanie implementacji

## Klasyfikacja Binarna



## Podsumowanie algorytmu SVM:

Implementacja SVM w najprostszej wersji okazała się dość niestabilnym klasyfikatorem dla zagadnienia binarnego. Dość duże oscylacje trafności SVM z własnej implementacji są przeciwieństwem dla niemalże stałej odpowiedzi implementacji z biblioteki R. Parametr C, kosztu, niemalże nie ma wpływu na Trafność dla SVM z biblioteki R.

Wyniki trafności dla obu klasyfikatorów nie są wybitnie wysokie, aczkolwiek pewniejszym będzie algorytm zaimplementowany w bibliotece R.

# Sieci Neuronowe (własna implementacja – 2 warstwy neuronów)

## Klasyfikacja Binarna

**nn_grid_bin = expand.grid(**

**h = list(data.frame(4,4), data.frame(5,5), data.frame(6,6), data.frame(7,7), data.frame(8,8)),**

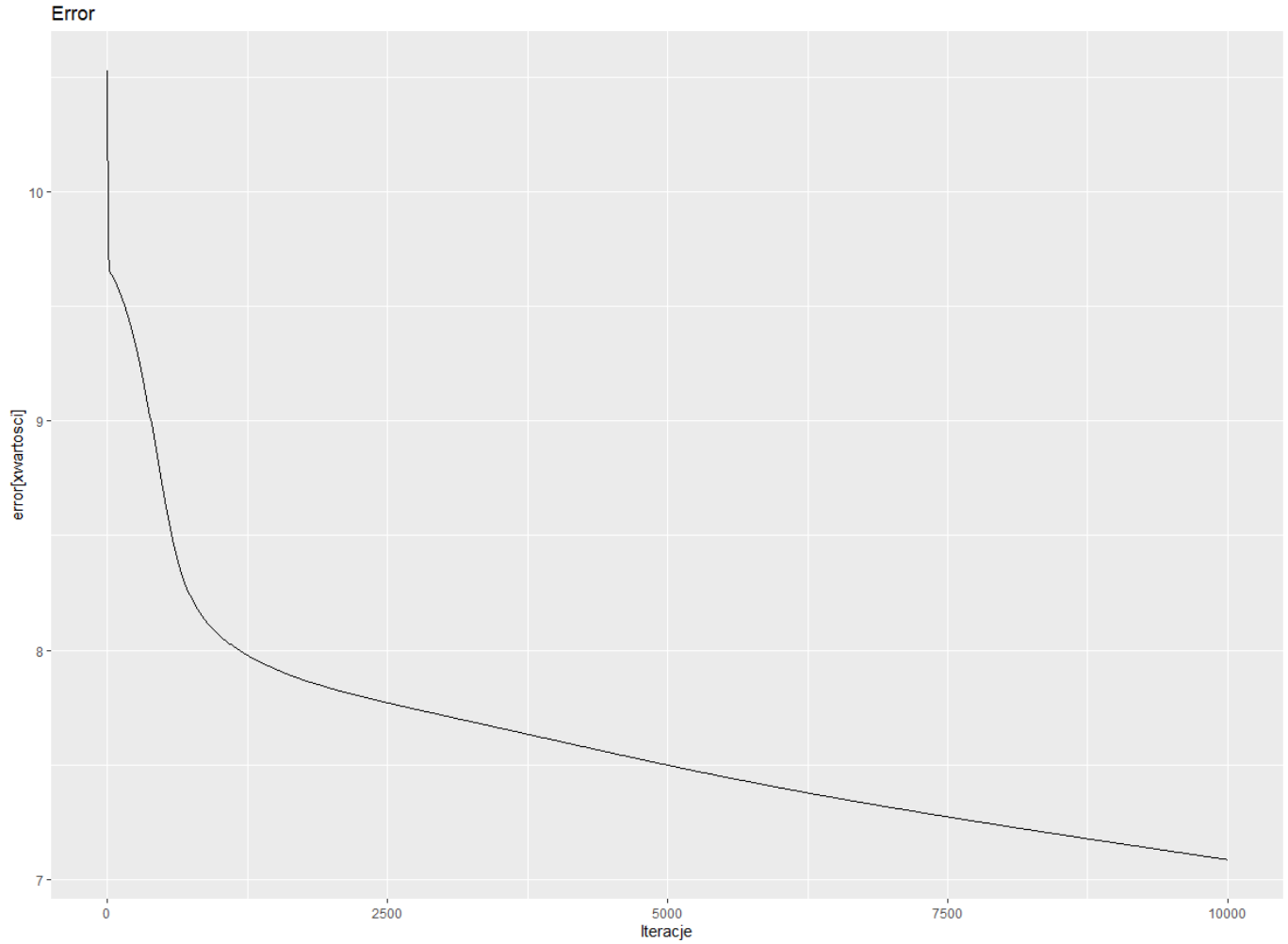
**lr = c(0.001),**

**iter = c(10000, 20000, 50000, 80000, 100000))**

| | h | lr | iter | Czulosc_TRAIN | Specyficznosc_TRAIN | Jakosc_TRAIN | Czulosc_TEST | Specyficznosc_TEST | Jakosc_TEST |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 4, 4 | 0.001 | 1e+04 | 1 | 0 | 0.553125 | 1 | 0 | 0.6625 |
| 2 | 5, 5 | 0.001 | 1e+04 | 1 | 0 | 0.562500 | 1 | 0 | 0.6250 |
| 3 | 6, 6 | 0.001 | 1e+04 | 1 | 0 | 0.600000 | 1 | 0 | 0.4750 |
| 4 | 7, 7 | 0.001 | 1e+04 | 1 | 0 | 0.596875 | 1 | 0 | 0.4875 |
| 5 | 8, 8 | 0.001 | 1e+04 | 1 | 0 | 0.593750 | 1 | 0 | 0.5000 |
| 6 | 4, 4 | 0.001 | 2e+04 | 1 | 0 | 0.568750 | 1 | 0 | 0.6000 |
| 7 | 5, 5 | 0.001 | 2e+04 | 1 | 0 | 0.562500 | 1 | 0 | 0.6250 |
| 8 | 6, 6 | 0.001 | 2e+04 | 1 | 0 | 0.600000 | 1 | 0 | 0.4750 |
| 9 | 7, 7 | 0.001 | 2e+04 | 1 | 0 | 0.596875 | 1 | 0 | 0.4875 |
| 10 | 8, 8 | 0.001 | 2e+04 | 1 | 0 | 0.593750 | 1 | 0 | 0.5000 |
| 11 | 4, 4 | 0.001 | 5e+04 | 1 | 0 | 0.568750 | 1 | 0 | 0.6000 |
| 12 | 5, 5 | 0.001 | 5e+04 | 1 | 0 | 0.562500 | 1 | 0 | 0.6250 |
| 13 | 6, 6 | 0.001 | 5e+04 | 1 | 0 | 0.600000 | 1 | 0 | 0.4750 |
| 14 | 7, 7 | 0.001 | 5e+04 | 1 | 0 | 0.596875 | 1 | 0 | 0.4875 |
| 15 | 8, 8 | 0.001 | 5e+04 | 1 | 0 | 0.593750 | 1 | 0 | 0.5000 |
| 16 | 4, 4 | 0.001 | 8e+04 | 1 | 0 | 0.568750 | 1 | 0 | 0.6000 |
| 17 | 5, 5 | 0.001 | 8e+04 | 1 | 0 | 0.562500 | 1 | 0 | 0.6250 |
| 18 | 6, 6 | 0.001 | 8e+04 | 1 | 0 | 0.600000 | 1 | 0 | 0.4750 |
| 19 | 7, 7 | 0.001 | 8e+04 | 1 | 0 | 0.596875 | 1 | 0 | 0.4875 |
| 20 | 8, 8 | 0.001 | 8e+04 | 1 | 0 | 0.593750 | 1 | 0 | 0.5000 |
| 21 | 4, 4 | 0.001 | 1e+05 | 1 | 0 | 0.568750 | 1 | 0 | 0.6000 |
| 22 | 5, 5 | 0.001 | 1e+05 | 1 | 0 | 0.562500 | 1 | 0 | 0.6250 |
| 23 | 6, 6 | 0.001 | 1e+05 | 1 | 0 | 0.600000 | 1 | 0 | 0.4750 |
| 24 | 7, 7 | 0.001 | 1e+05 | 1 | 0 | 0.596875 | 1 | 0 | 0.4875 |
| 25 | 8, 8 | 0.001 | 1e+05 | 1 | 0 | 0.593750 | 1 | 0 | 0.5000 |

# Sieci Neuronowe (własna implementacja – 2 warstwy neuronów)

## Wykres Error podczas uczenia na całym zbiorze do klasyfikacji Binarnej



# Sieci Neuronowe (własna implementacja – 2 warstwy neuronów)

## Klasyfikacja Wieloklasowa

**nn_grid_multi = expand.grid(**

**h = list(data.frame(4,4), data.frame(5,5), data.frame(6,6), data.frame(7,7), data.frame(8,8)),**

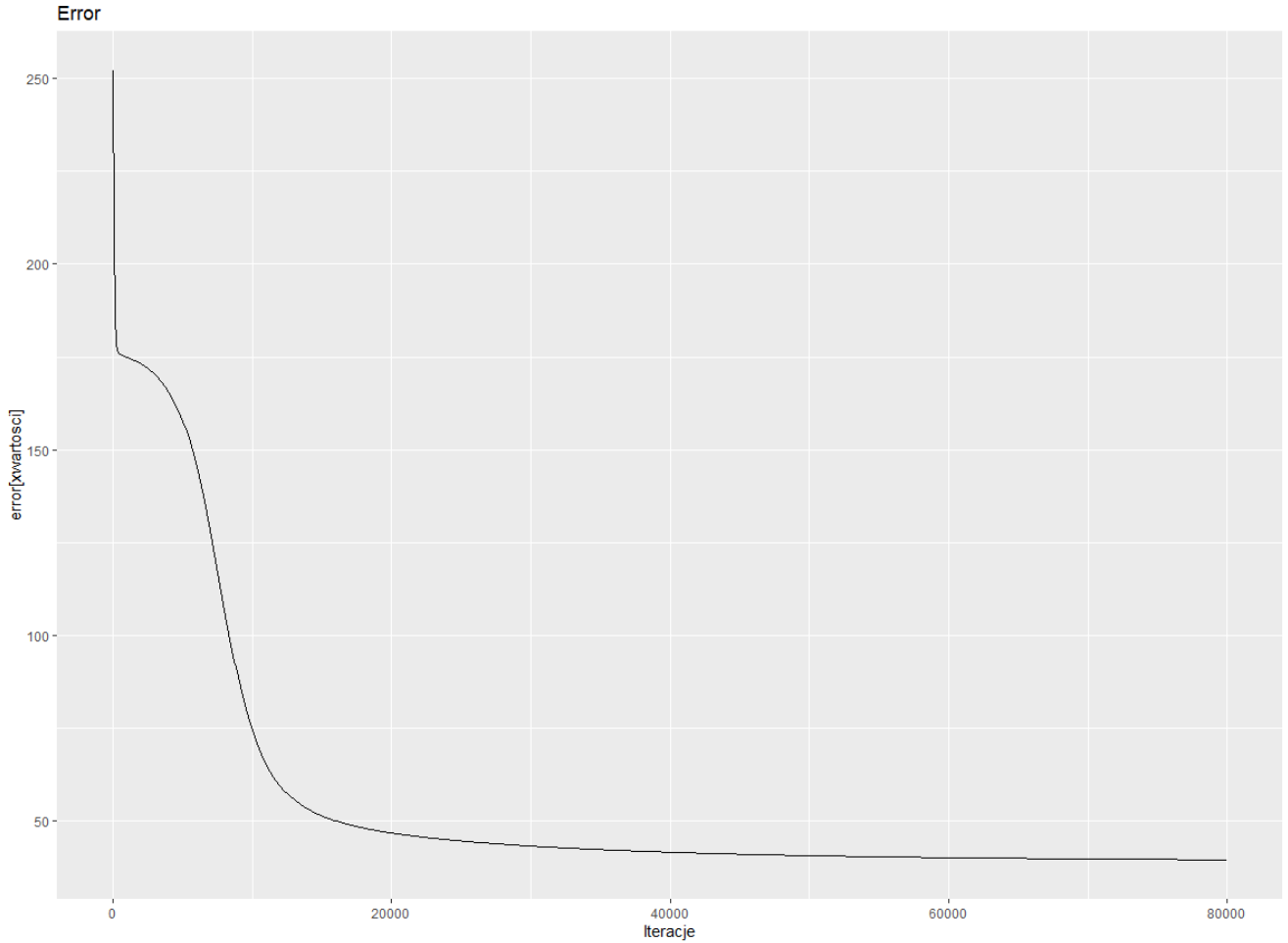
**lr = c(0.001),**

**iter = c(10000, 20000, 50000, 80000, 100000))**

| | h | lr | iter | Jakosc_TRAIN | Jakosc_TEST |
|---|---|---|---|---|---|
| 1 | 4, 4 | 0.001 | 1e+04 | 0.4620 | 0.4356 |
| 2 | 5, 5 | 0.001 | 1e+04 | 0.7004 | 0.4340 |
| 3 | 6, 6 | 0.001 | 1e+04 | 0.6288 | 0.4492 |
| 4 | 7, 7 | 0.001 | 1e+04 | 0.2592 | 0.4500 |
| 5 | 8, 8 | 0.001 | 1e+04 | 0.2728 | 0.4360 |
| 6 | 4, 4 | 0.001 | 2e+04 | 0.4676 | 0.4336 |
| 7 | 5, 5 | 0.001 | 2e+04 | 0.7252 | 0.4284 |
| 8 | 6, 6 | 0.001 | 2e+04 | 0.7152 | 0.4560 |
| 9 | 7, 7 | 0.001 | 2e+04 | 0.4872 | 0.4408 |
| 10 | 8, 8 | 0.001 | 2e+04 | 0.4268 | 0.4556 |
| 11 | 4, 4 | 0.001 | 5e+04 | 0.5920 | 0.4420 |
| 12 | 5, 5 | 0.001 | 5e+04 | 0.7924 | 0.4280 |
| 13 | 6, 6 | 0.001 | 5e+04 | 0.8196 | 0.4596 |
| 14 | 7, 7 | 0.001 | 5e+04 | 0.7472 | 0.4496 |
| 15 | 8, 8 | 0.001 | 5e+04 | 0.7348 | 0.4536 |
| 16 | 4, 4 | 0.001 | 8e+04 | 0.7360 | 0.4436 |
| 17 | 5, 5 | 0.001 | 8e+04 | 0.8588 | 0.4452 |
| 18 | 6, 6 | 0.001 | 8e+04 | 0.8720 | 0.4736 |
| 19 | 7, 7 | 0.001 | 8e+04 | 0.8344 | 0.4524 |
| 20 | 8, 8 | 0.001 | 8e+04 | 0.7824 | 0.4564 |
| 21 | 4, 4 | 0.001 | 1e+05 | 0.8060 | 0.4368 |
| 22 | 5, 5 | 0.001 | 1e+05 | 0.8796 | 0.4452 |
| 23 | 6, 6 | 0.001 | 1e+05 | 0.8772 | 0.4716 |
| 24 | 7, 7 | 0.001 | 1e+05 | 0.8740 | 0.4652 |
| 25 | 8, 8 | 0.001 | 1e+05 | 0.8328 | 0.4580 |

# Sieci Neuronowe (własna implementacja – 2 warstwy neuronów)

## Wykres Error podczas uczenia na zbiorze do klasyfikacji Wieloklasowej



# Sieci Neuronowe (własna implementacja – 2 warstwy neuronów)

## Regresja

**nn_grid_multi = expand.grid(**

**h = list(data.frame(4,4), data.frame(5,5), data.frame(6,6), data.frame(7,7), data.frame(8,8)),**

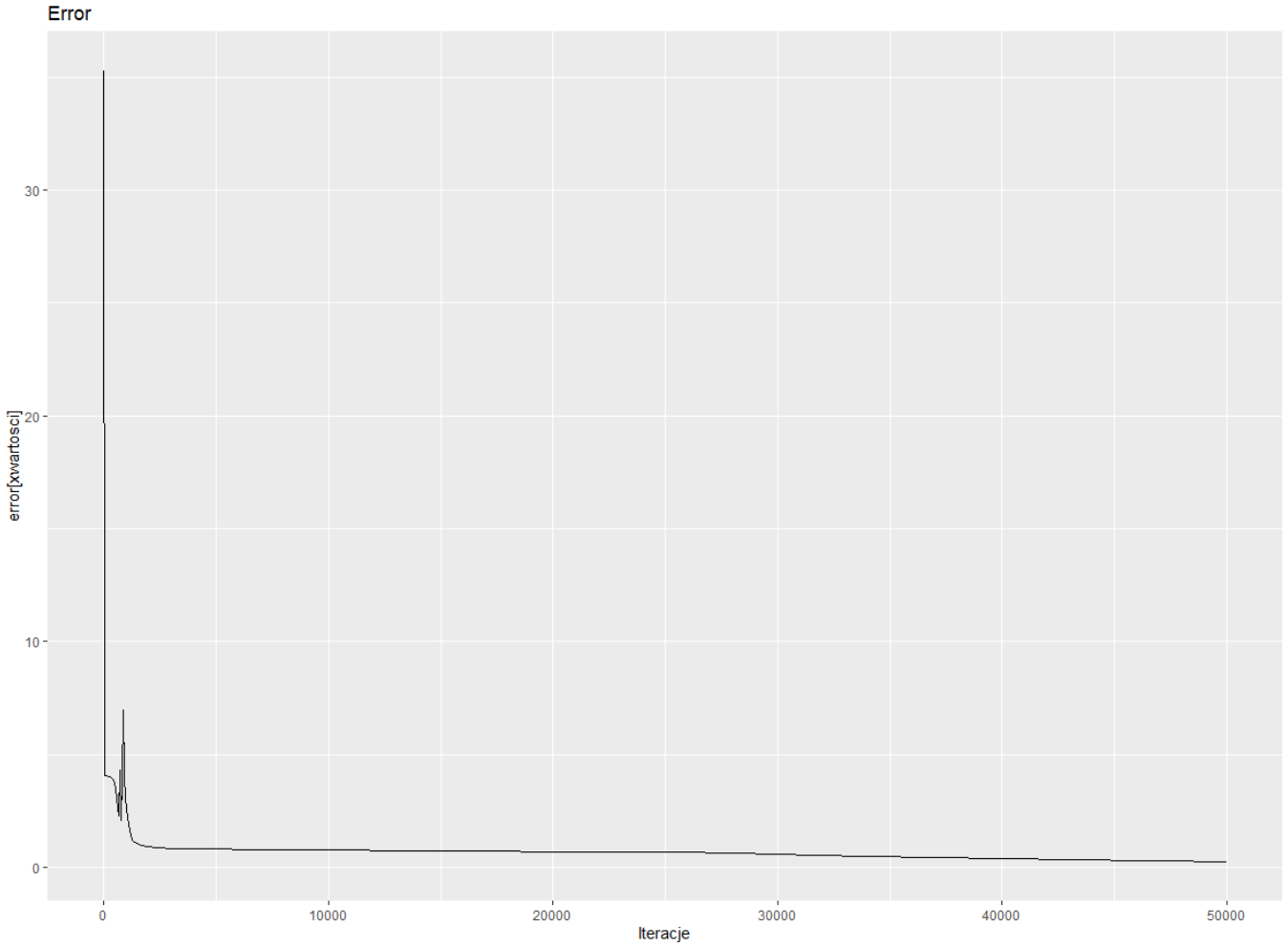
**lr = c(0.001),**

**iter = c(10000, 20000, 50000, 80000, 100000))**

| | h | lr | iter | MAE_TRAIN | MSE_TRAIN | MAPE_TRAIN | MAE_TEST | MSE_TEST | MAPE_TEST |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 4, 4 | 0.001 | 1e+04 | 0.6428503 | 0.7639935 | 0.7400306 | 0.5812200 | 0.5934162 | 0.7565529 |
| 2 | 5, 5 | 0.001 | 1e+04 | 0.5805535 | 0.6680637 | 0.6472925 | 0.5916473 | 0.5929658 | 0.7695442 |
| 3 | 6, 6 | 0.001 | 1e+04 | 0.6210131 | 0.7177338 | 0.7467128 | 0.7381840 | 0.9020054 | 0.7082569 |
| 4 | 7, 7 | 0.001 | 1e+04 | 0.5826569 | 0.6196749 | 0.7182905 | 0.7635898 | 1.1387122 | 0.7147901 |
| 5 | 8, 8 | 0.001 | 1e+04 | 0.5104059 | 0.5025658 | 0.6252082 | 0.7188911 | 1.1681766 | 0.5839337 |
| 6 | 4, 4 | 0.001 | 2e+04 | 0.4711496 | 0.5520885 | 0.4672651 | 0.3974458 | 0.4082859 | 0.4344668 |
| 7 | 5, 5 | 0.001 | 2e+04 | 0.4839927 | 0.5657735 | 0.4941931 | 0.4729905 | 0.4602828 | 0.5740935 |
| 8 | 6, 6 | 0.001 | 2e+04 | 0.4780102 | 0.5514039 | 0.5272130 | 0.6350459 | 0.7870918 | 0.5405550 |
| 9 | 7, 7 | 0.001 | 2e+04 | 0.4473539 | 0.4630616 | 0.4971654 | 0.5836559 | 0.9093017 | 0.4673589 |
| 10 | 8, 8 | 0.001 | 2e+04 | 0.3955131 | 0.3720087 | 0.4280549 | 0.5557748 | 0.9866210 | 0.3419601 |
| 11 | 4, 4 | 0.001 | 5e+04 | 0.4078735 | 0.4824849 | 0.3681130 | 0.3534728 | 0.3625822 | 0.3540378 |
| 12 | 5, 5 | 0.001 | 5e+04 | 0.4252319 | 0.4809942 | 0.4155773 | 0.4100868 | 0.3391847 | 0.4879808 |
| 13 | 6, 6 | 0.001 | 5e+04 | 0.4018041 | 0.4509018 | 0.3982552 | 0.5751574 | 0.6936388 | 0.4349889 |
| 14 | 7, 7 | 0.001 | 5e+04 | 0.3842785 | 0.3685915 | 0.3987115 | 0.4905616 | 0.8387685 | 0.3462382 |
| 15 | 8, 8 | 0.001 | 5e+04 | 0.3540590 | 0.3282008 | 0.3622393 | 0.5259005 | 1.0041600 | 0.2839828 |
| 16 | 4, 4 | 0.001 | 8e+04 | 0.3932981 | 0.4659471 | 0.3506859 | 0.3429728 | 0.3558639 | 0.3407892 |
| 17 | 5, 5 | 0.001 | 8e+04 | 0.3681511 | 0.3341335 | 0.3865401 | 0.3650761 | 0.2633269 | 0.4119247 |
| 18 | 6, 6 | 0.001 | 8e+04 | 0.3606656 | 0.3700510 | 0.3642664 | 0.5188544 | 0.5796434 | 0.3844382 |
| 19 | 7, 7 | 0.001 | 8e+04 | 0.3523900 | 0.3312913 | 0.3532937 | 0.4539211 | 0.7907191 | 0.3134736 |
| 20 | 8, 8 | 0.001 | 8e+04 | 0.3362661 | 0.3085609 | 0.3399491 | 0.5042360 | 0.9638853 | 0.2726240 |
| 21 | 4, 4 | 0.001 | 1e+05 | 0.3852809 | 0.4491291 | 0.3450256 | 0.3370940 | 0.3493786 | 0.3398211 |
| 22 | 5, 5 | 0.001 | 1e+05 | 0.3526641 | 0.3074505 | 0.3653131 | 0.3420835 | 0.2356359 | 0.3757700 |
| 23 | 6, 6 | 0.001 | 1e+05 | 0.3314293 | 0.2921592 | 0.3578711 | 0.4469096 | 0.4436799 | 0.3680586 |
| 24 | 7, 7 | 0.001 | 1e+05 | 0.3373189 | 0.3115149 | 0.3397765 | 0.4319811 | 0.7433113 | 0.2882737 |
| 25 | 8, 8 | 0.001 | 1e+05 | 0.3260855 | 0.2875617 | 0.3313991 | 0.4775881 | 0.8735837 | 0.2669550 |

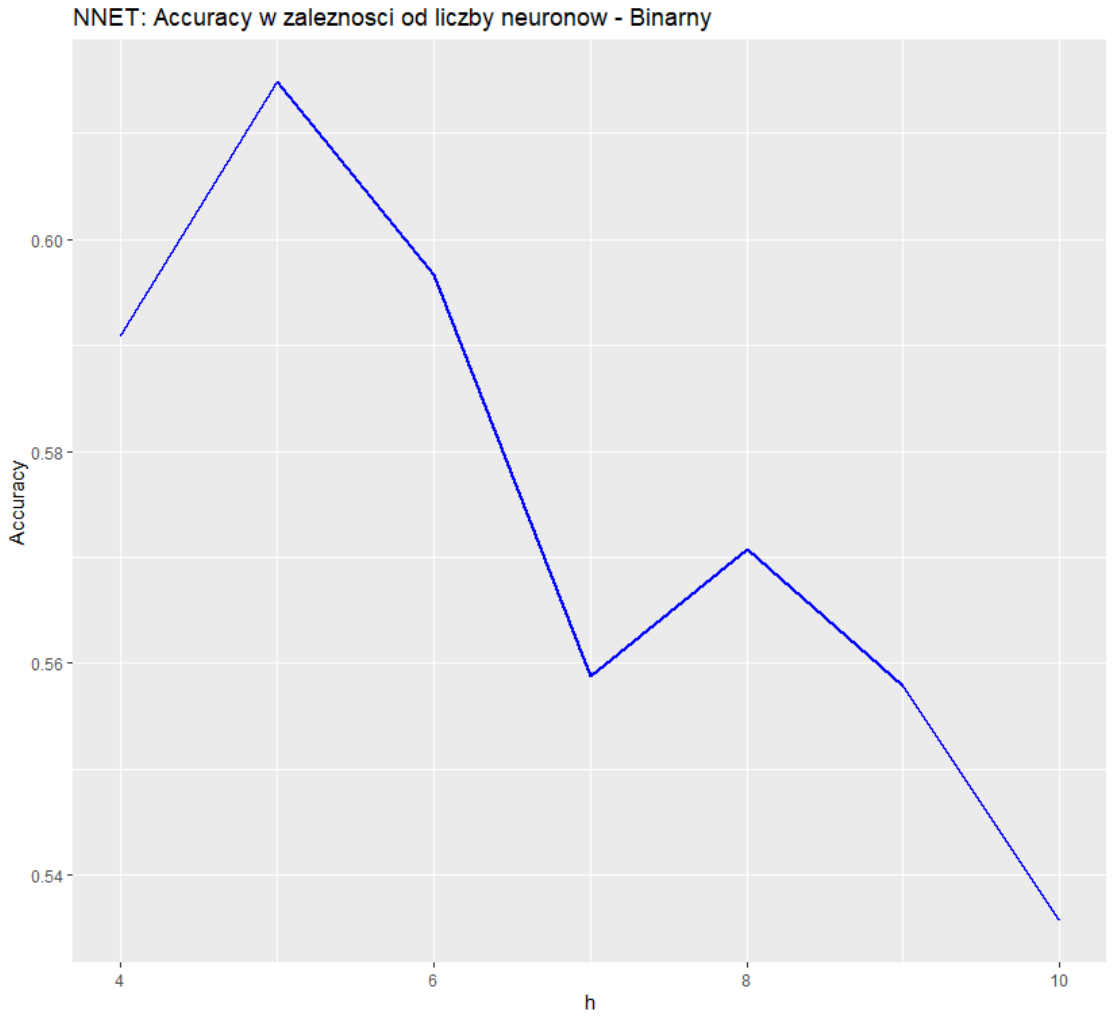
# Sieci Neuronowe (własna implementacja – 2 warstwy neuronów)

## Wykres Error podczas uczenia na zbiorze do Regresji



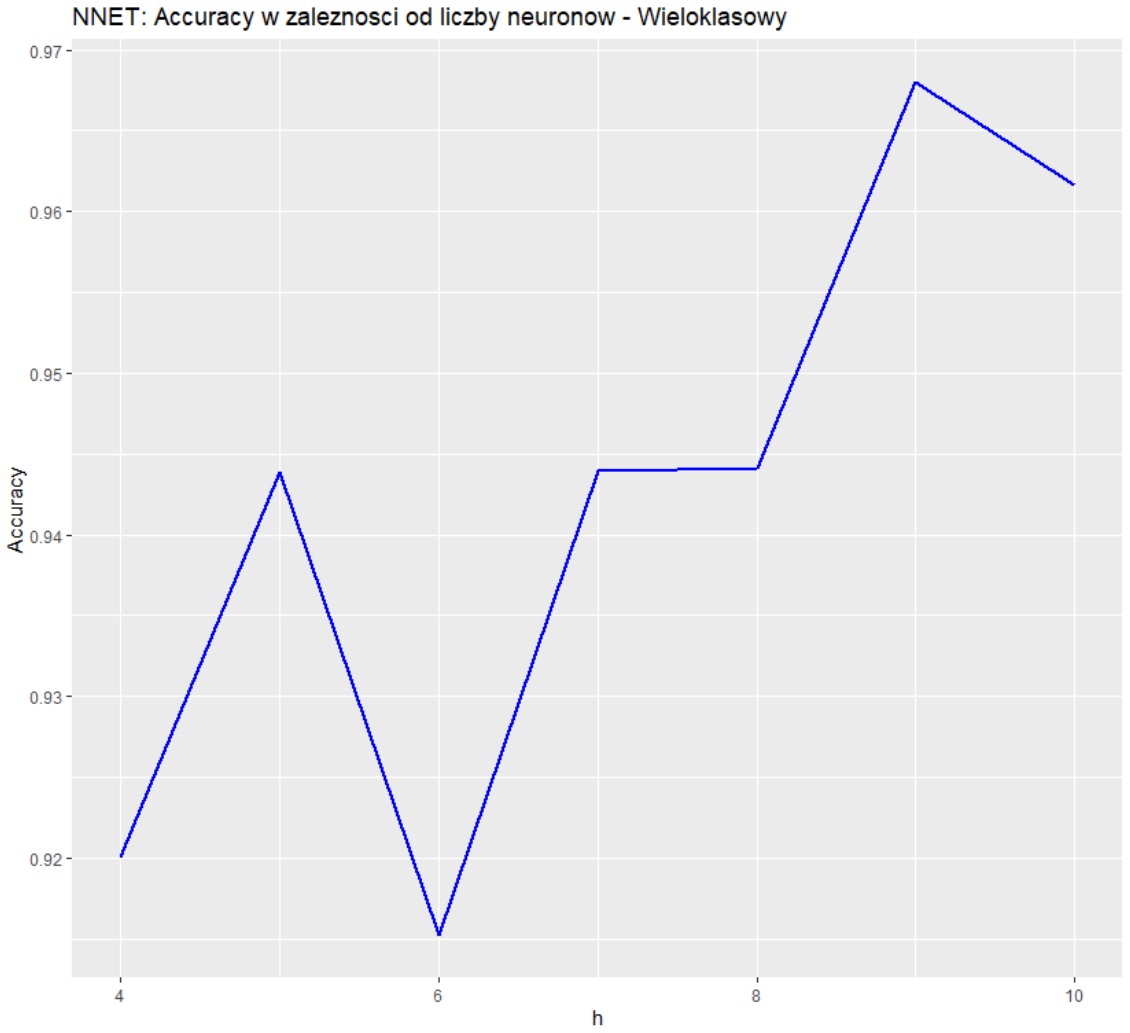
# Sieci Neuronowe (NNET)

## Klasyfikacja Binarna



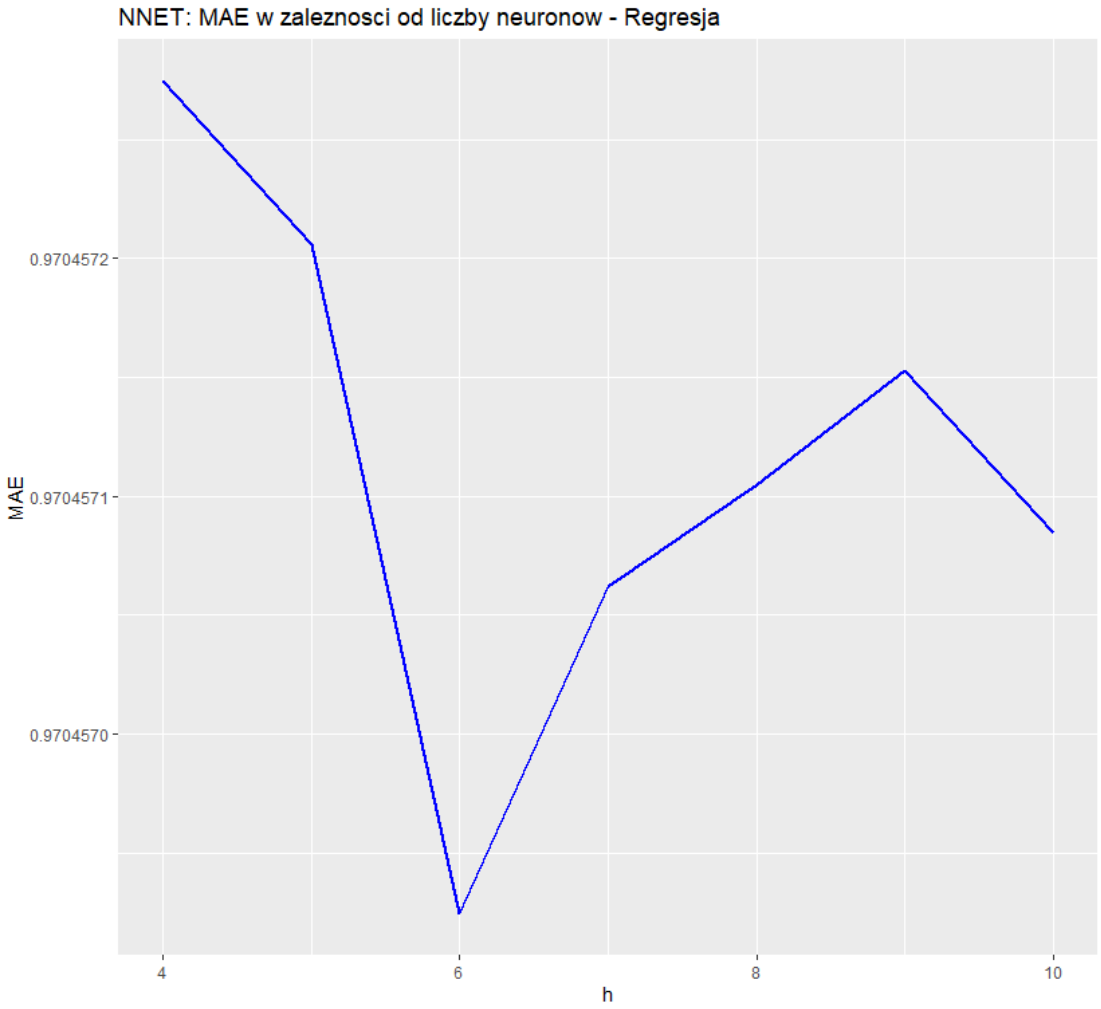
# Sieci Neuronowe (NNET)

## Klasyfikacja Wieloklasowa



# Sieci Neuronowe (NNET)

## Regresja



## Podsumowanie algorytmu Sieci Neuronowych:

Algorytmy na bazie sieci neuronowych nie radziły sobie najlepiej w zagadnieniu klasyfikacji binarnej. Dla obu implementacji, własnej jak i *nnet*, trafność klasyfikacji binarnej była najgorsza ze wszystkich klasyfikatorów. Należy pamiętać, że zbiór danych nie był duży, a takie algorytmy jak sieci neuronowe potrzebują dużo danych aby w pełni wykorzystać swój potencjał.

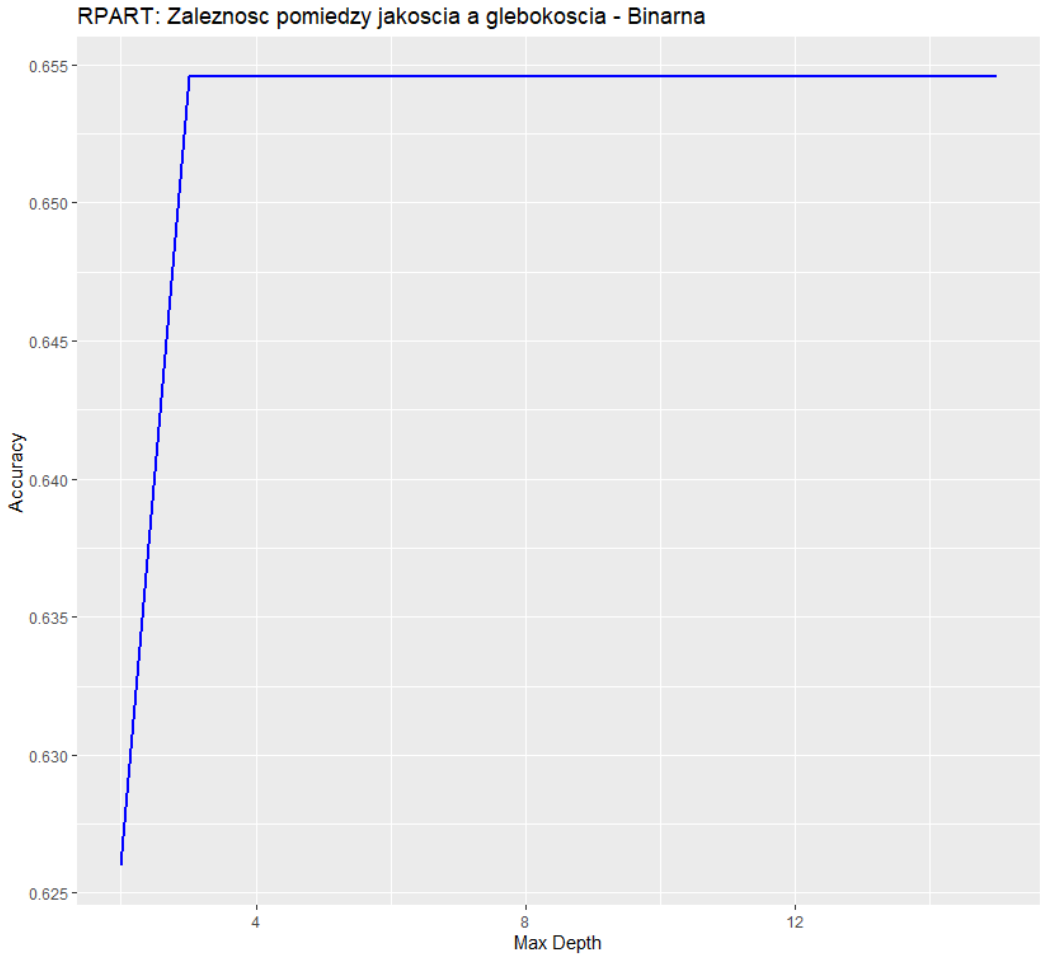
Klasyfikacja na wielu klasach była zdecydowanie lepsza, przynajmniej dla implementacji z biblioteki R. Wysoki wynik trafności na poziomie 0.9679 świadczy o dobrze nauczonych klasyfikatorze. Implementacja własna tego algorytmu niestety nie poradziła sobie z tym zagadnieniem. Na taki stan rzeczy mogło się nałożyć kilka czynników, np. wykorzystanie innej funkcji aktywacji, lub inna ilość neuronów w warstwie czy samych warstw neuronów.

Regresja dla sieci jest za to bardzo zróżnicowana – własna implementacja posiada MAE na średnim poziomie ze wszystkich algorytmów, gdzie sieci na bazie *nnet* wykazują najwyższe MAE.

Dla sieci neuronowych własnej implementacji, można zobaczyć różną charakterystykę błędu w zależności od iteracji uczenia. Na niektórych wykresach widać punkty przegięcia, które świadczą o nagłej zmianie wartości wag pozwalającej na lepsze ich dopasowanie względem wartości oczekiwanych. W niektórych przypadkach, z testów wynikało, że podniesienie ilości iteracji z 80 tys na 100 tys pozwoliło na zmniejszenie MAE o kolejne wartości.

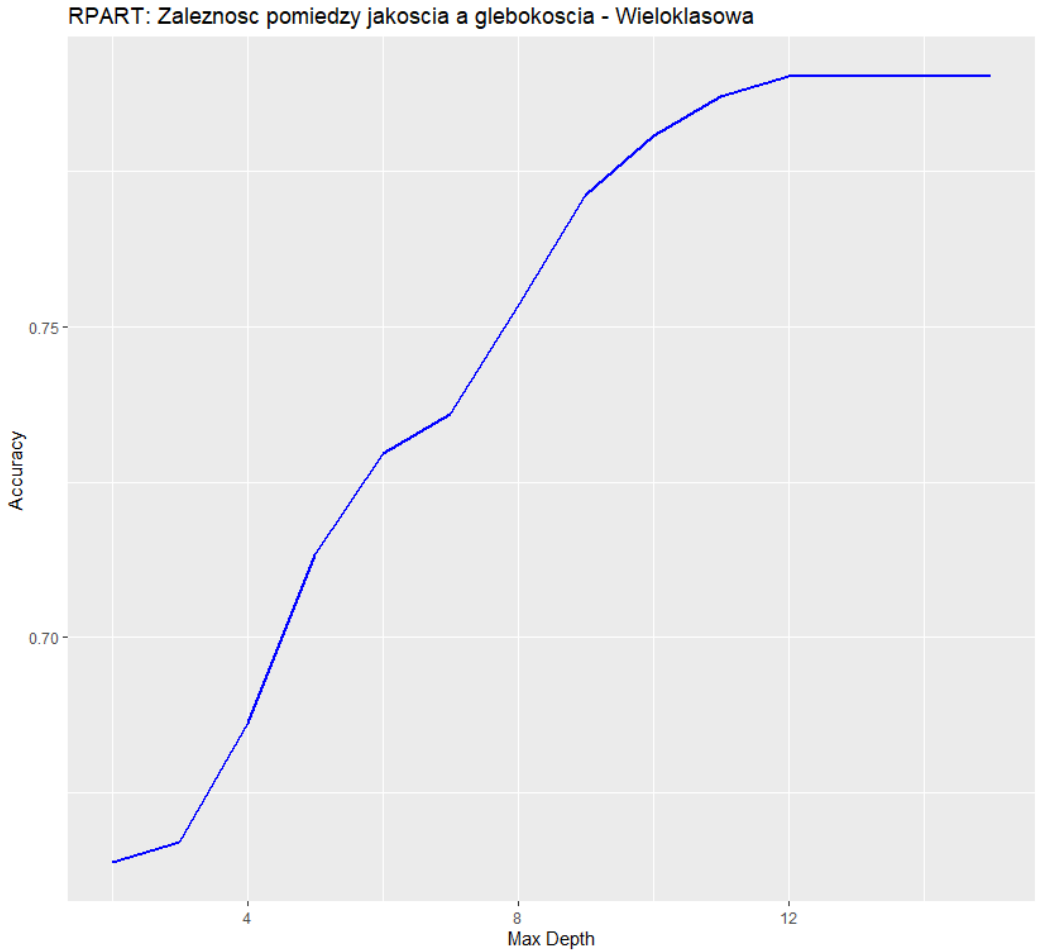
# Drzewa Decyzyjne (Rpart)

## Klasyfikacja Binarna



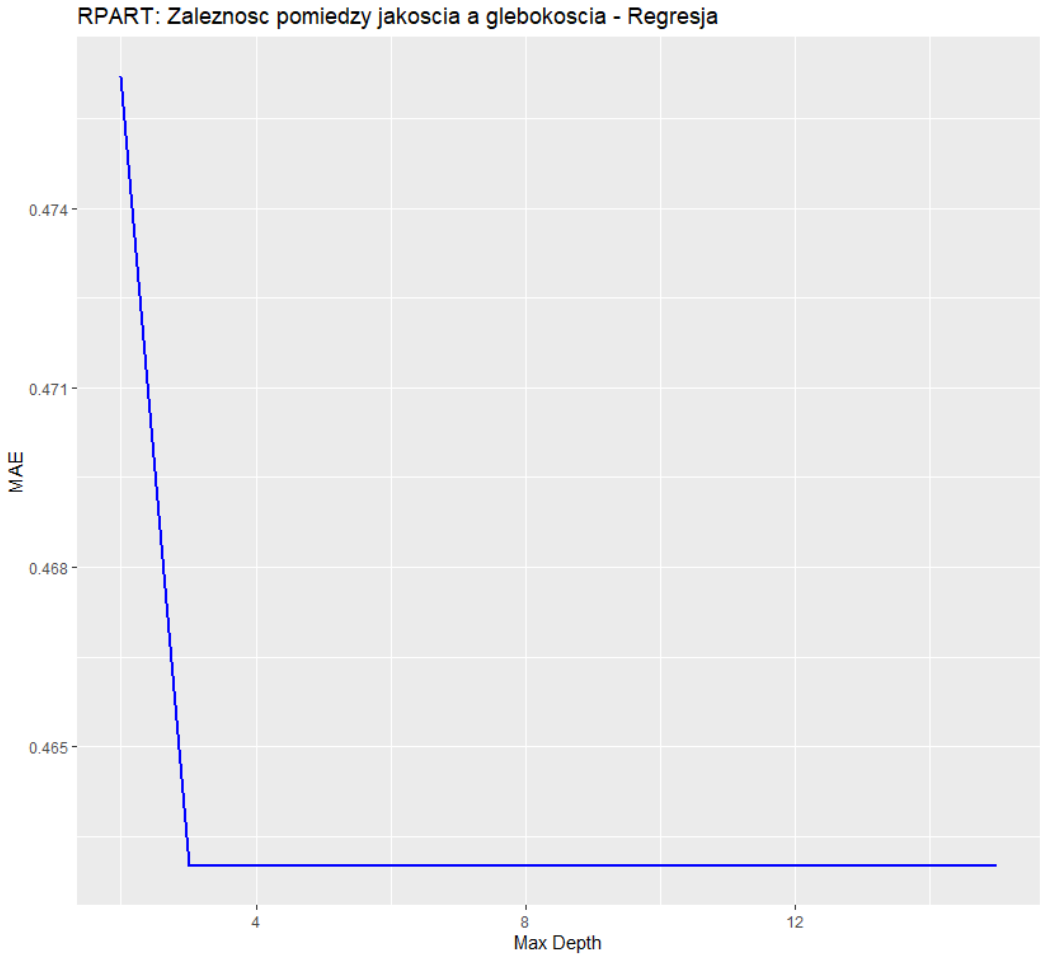
# Drzewa Decyzyjne (Rpart)

## Klasyfikacja Wieloklasowa



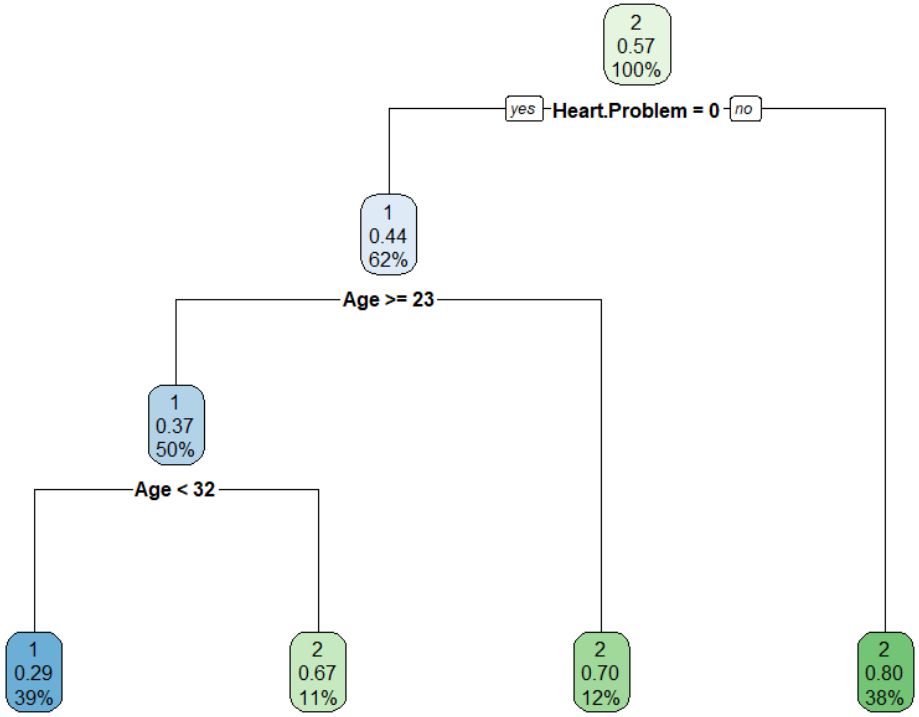
# Drzewa Decyzyjne (Rpart)

## Regresja



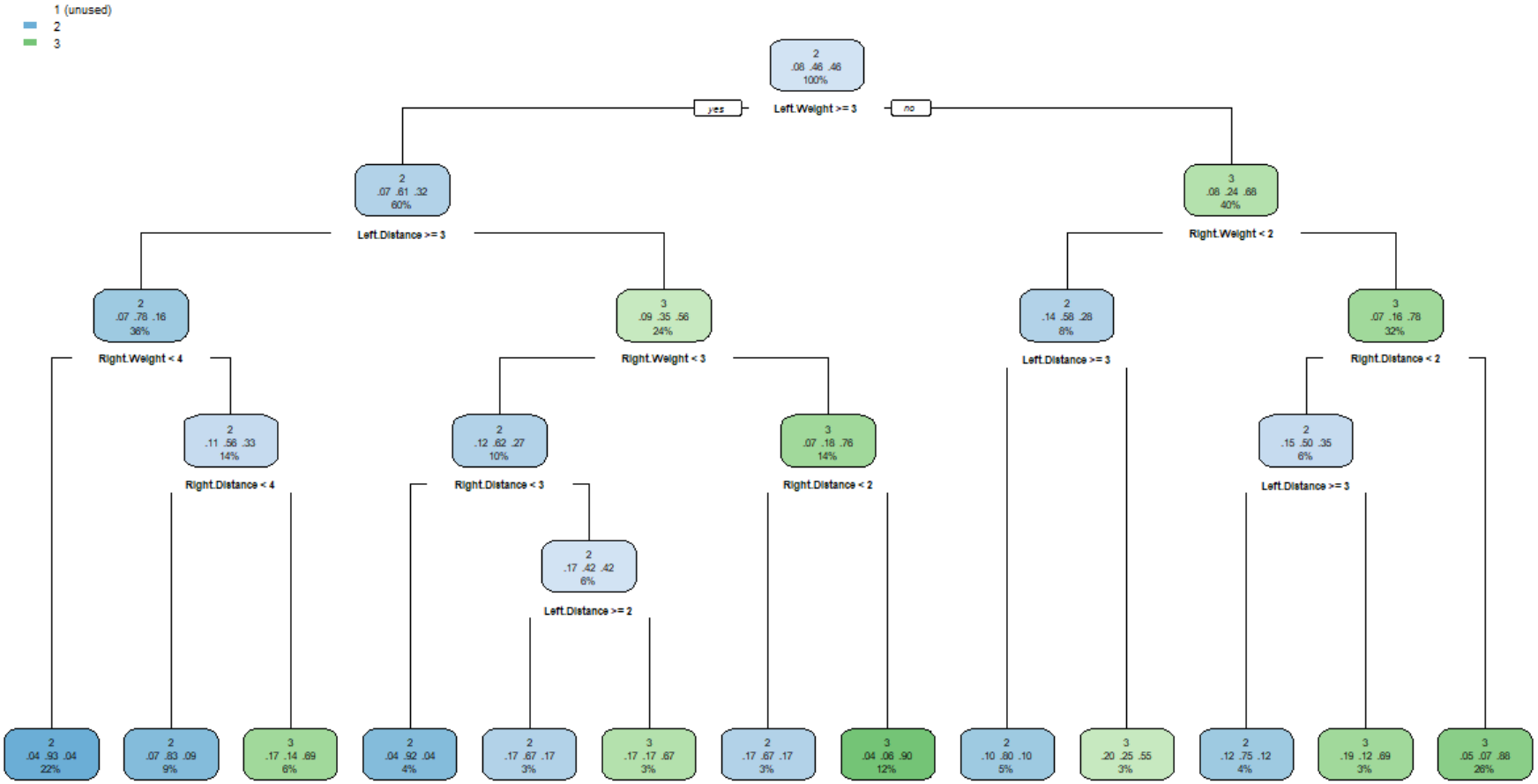
# Drzewa Decyzyjne (Rpart) – Wizualizacja drzewa

## Klasyfikacja Binarna



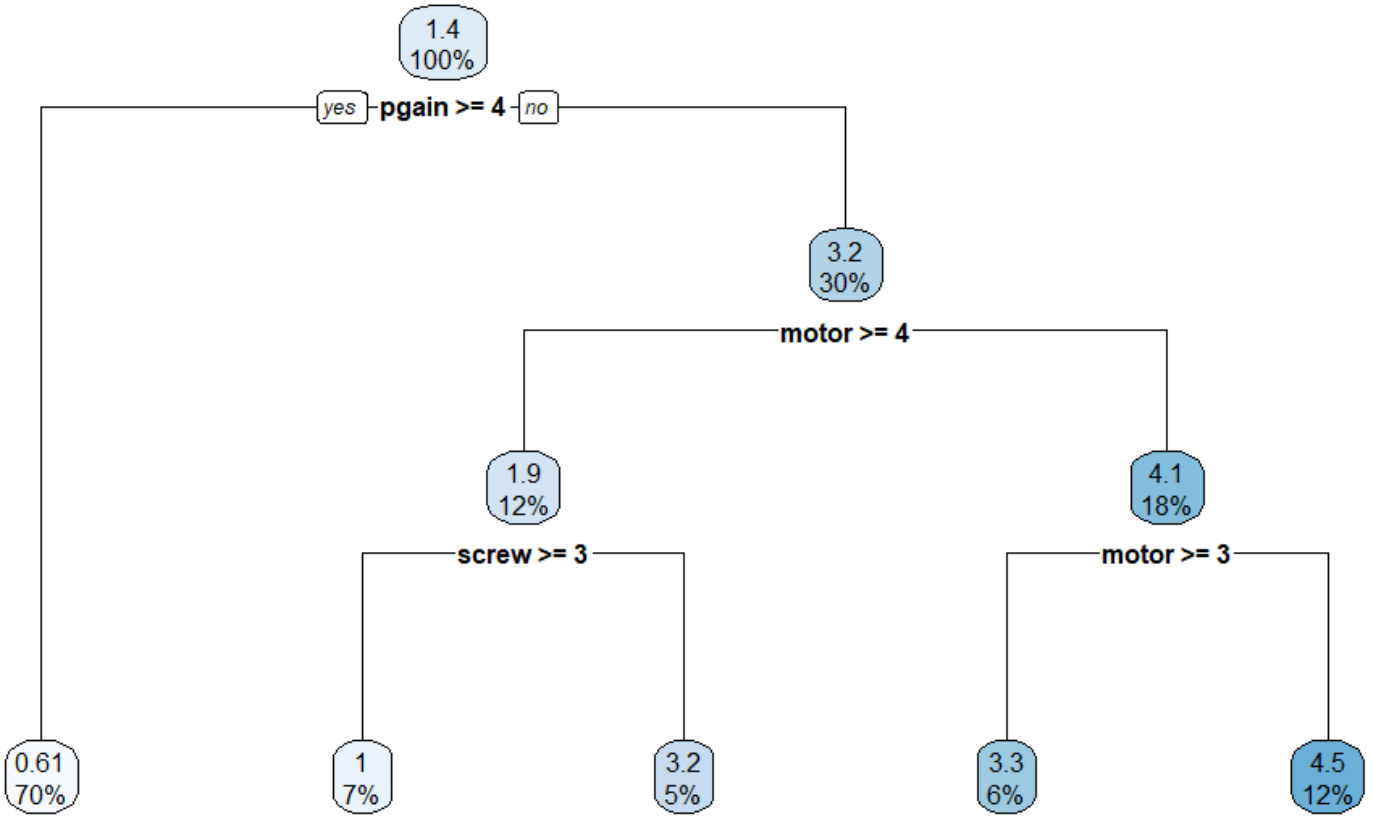
# Drzewa Decyzyjne (Rpart) – Wizualizacja drzewa

## Klasyfikacja Wieloklasowa



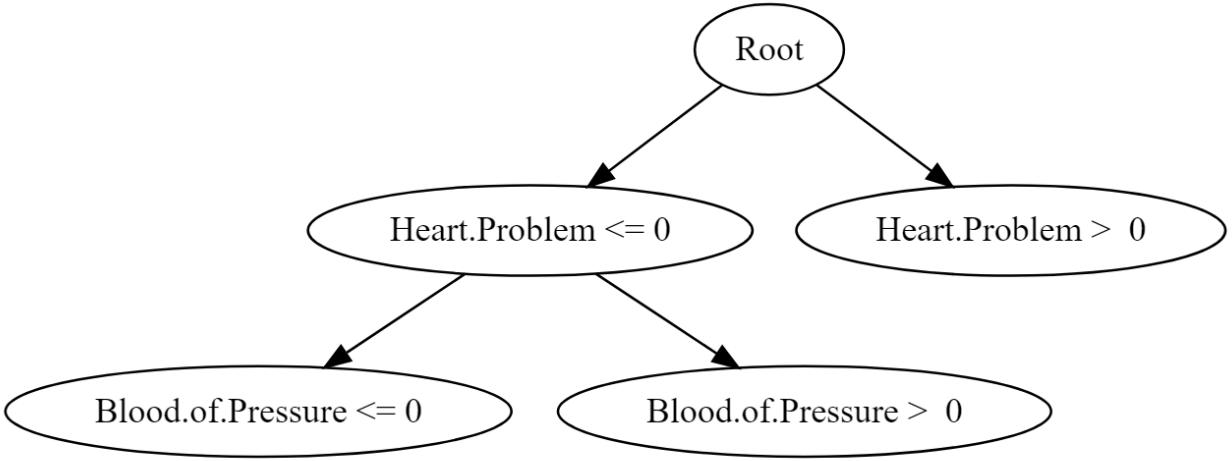
# Drzewa Decyzyjne (Rpart) – Wizualizacja drzewa

## Regresja



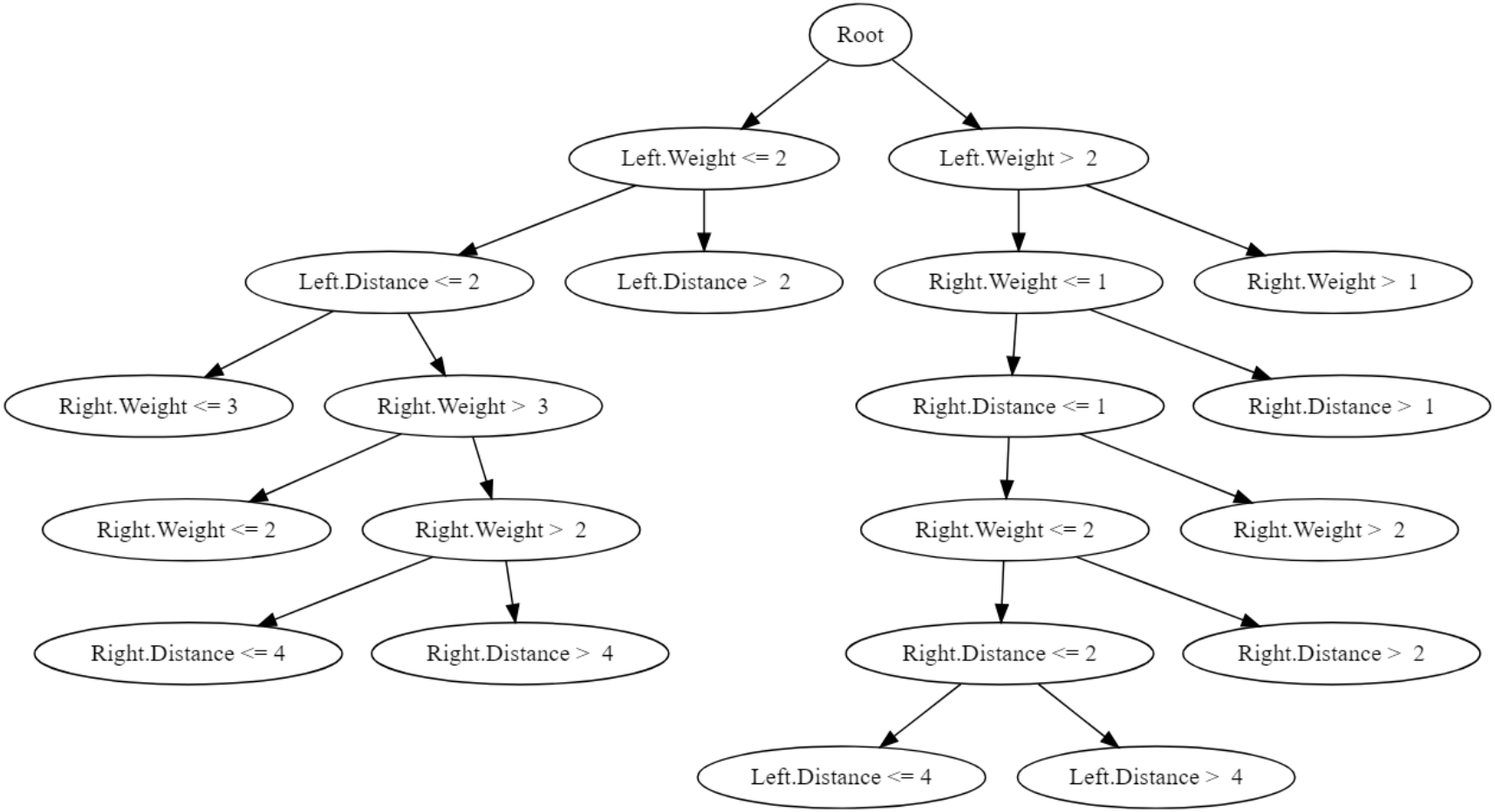
# Drzewa Decyzyjne (własna implementacja) – Wizualizacja drzewa

**Klasyfikacja Binarna**



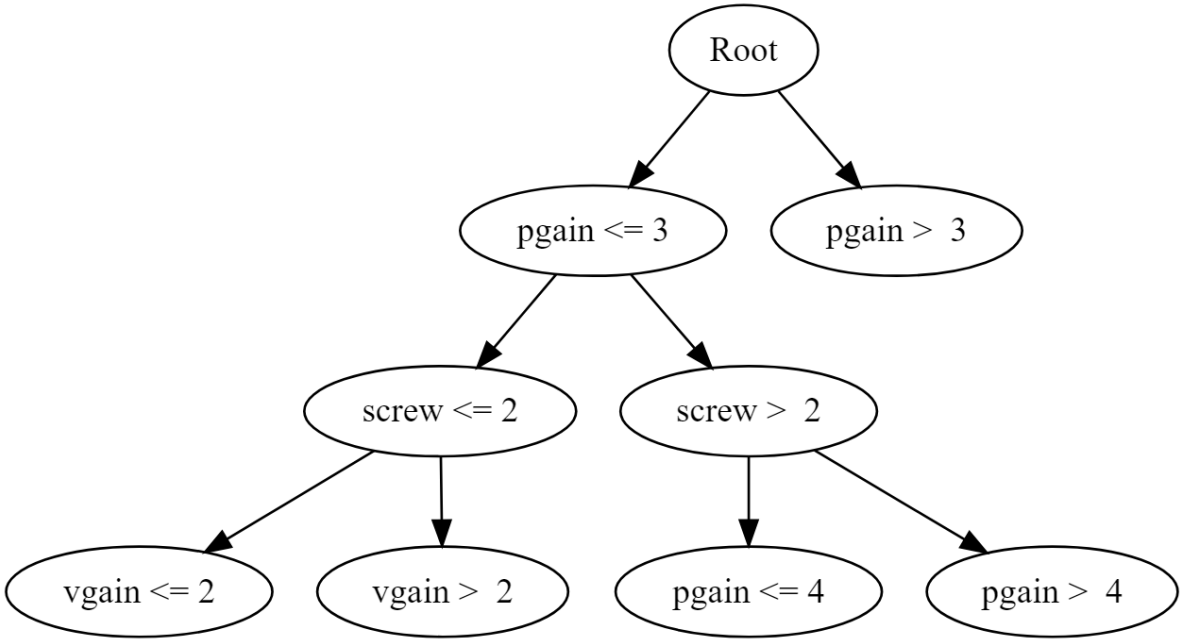
# Drzewa Decyzyjne (własna implementacja) – Wizualizacja drzewa

## Klasyfikacja Wieloklasowa



# Drzewa Decyzyjne (własna implementacja) – Wizualizacja drzewa

**Regresja**



## Podsumowanie algorytmu Drzew Decyzyjnych:

Ze względu na brak funkcji do predykcji na własnej implementacji drzew decyzyjnych, skupiono się na przeglądzie algorytmów opartych o bibliotekę rpart oraz na wizualizacji drzew decyzyjnych.

Sprawdzenie zależności wpływu głębokości drzewa na trafność oraz błąd MAE wykazało, że tylko przy klasyfikacji wieloklasowej kolejne głębokości drzewa mają wpływ na trafność. W pozostałych przypadkach dla głębokości drzewa równej 3, drzewo decyzyjne osiągało stabilną trafność czy błąd MAE.

Dla najlepszych parametrów drzewa wykonano wizualizację takiego drzewa przy pomocy biblioteki rpart.plot. Pozwala to lepiej zrozumieć idee i działanie takiego drzewa. Dla klasyfikacji wieloklasowej widać, że drzewo jest dość rozbudowane i głębokość drzewa miałaby duży wpływ na odpowiednią predykcję.

Dla najlepszych parametrów drzew z biblioteki rpart wykonano również nauczanie drzew własnej implementacji oraz wizualizację takich struktur. Widać, po wizualizacji jednak, że struktury zaimplementowane ręcznie nie pokrywają się z tymi z biblioteki rpart. Można jedynie przypuszczać, że takie drzewa decyzyjne nie byłyby idealnymi klasyfikatorami.

# Podsumowanie

Podsumowując poszczególne fragmenty związane z różnymi klasyfikatorami i ich działaniem warto spojrzeć na tabele z najlepszymi modelami dla poszczególnych algorytmów.
Przy klasyfikacji binarnej najlepszym algorytmem okazał się *k Najbliższych Sąsiadów* – z trafnością na poziomie 0.8125 był zdecydowanie najlepszym klasyfikatorem ze wszystkich sprawdzonych.
Sieci Neuronowe z biblioteki *nnet* uzyskały najlepszy wynik trafności jeśli chodzi o klasyfikację wieloklasową – 0.9679. Tuż za nimi był algorytm *k-nn*, który w obu implementacjach uzyskał podobne wyniki.
Przy zagadnienie regresji zdecydowanie górowały algorytmy k-nn, osiągając w obu implementacjach bardzo podobne wyniki przy tym samym parametrze k = 2.

Ręczna implementacja algorytmów pozwala poznać różnorodność i wiele możliwości zróżnicowania kodu odpowiedzialnego ideowo za ten sam algorytm. Każdą z tych metod można urozmaicić o kolejne pomysły przyśpieszając czy zwiększając skuteczność danych algorytmów.