Raport 4
Zaawansowane algorytmy klasyfikacji
Analiza skupień

Romana Żmuda
249706
Adrian Kit
249746

15 czerwca 2020

Spis treści

1 Zadanie 1 - Zaawansowane metody klasyfikacji

W tym sprawozdaniu kontynuujemy rozważania z Raportu 3 dotyczące danych o winach. Tym tazem skorzystamy z bardziej zaawansowanych metod, takich jak lasy losowe czy algorytmy bagging lub boosting. Porównamy je z metodami użytymi w poprzednim raporcie i na tej podstawie wybierzemy najlepszą z nich.

1.1 Rodziny klasyfikatorów - bagging, boosting, random forest

1.2 Rodziny klasyfikatorów

W tym zadaniu omówimy działanie algorytmów *bagging*, *AdaBoost* (jako wariant algorytmu *boosting*) oraz *random forest* (lasy losowe). Wykorzystamy dane *wine* z pakietu "HDClassif". Celem będzie zbadanie dokładności oraz porównanie ich z drzewem losowym z poprzedniego raportu oraz między sobą tak, by wybrać ten najefektywniejszy.

By zachować pełną analogię względem poprzedniego raportu analizę przeprowadzimy z podziałem na zbiór uczący i testowy. Dodatkowo, porównamy wyniki otrzymane przy badaniu całego zbioru danych względem zmiennych, które wybralismy jako te o najlepszej zdolności dyskryminacyjnej, czyli: Phenols i Flavanoids.

Jedyną zmianą będzie ustawienie ziarna generatora, dzięki czemu otrzymamy stałe wyniki. Musimy zatem na nowo przeanalizować algorytm drzewa klasyfikacyjnego.

1.2.1 Pojedyncze drzewo klasyfikacyjne

Cały zbiór danych

- Grupa ucząca

  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	35	0	0
2	0	46	1
3	0	0	36

Tabela 1: Macierz pomyłek - drzewo klasyfikacyjne, grupa ucząca

  Błąd klasyfikacyjny:

```
## [1] 0.008474576
```

- Grupa testowa

  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	24	0	0
2	2	20	2
3	0	0	12

Tabela 2: Macierz pomyłek - drzewo klasyfikacyjne, grupa testowa

  Błąd klasyfikacyjny:

```
## [1] 0.06666667
```

Zmienne o najlepszej zdolności dyskryminacyjnej

- Grupa ucząca

  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	30	5	0
2	7	34	6
3	0	1	35

Tabela 3: Macierz pomyłek - drzewo klasyfikacyjne, grupa ucząca

Błąd klasyfikacyjny:

```
## [1] 0.1610169
```

- Grupa testowa

  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	19	5	0
2	7	12	5
3	0	0	12

Tabela 4: Macierz pomyłek - drzewo klasyfikacyjne, grupa testowa

Błąd klasyfikacyjny:

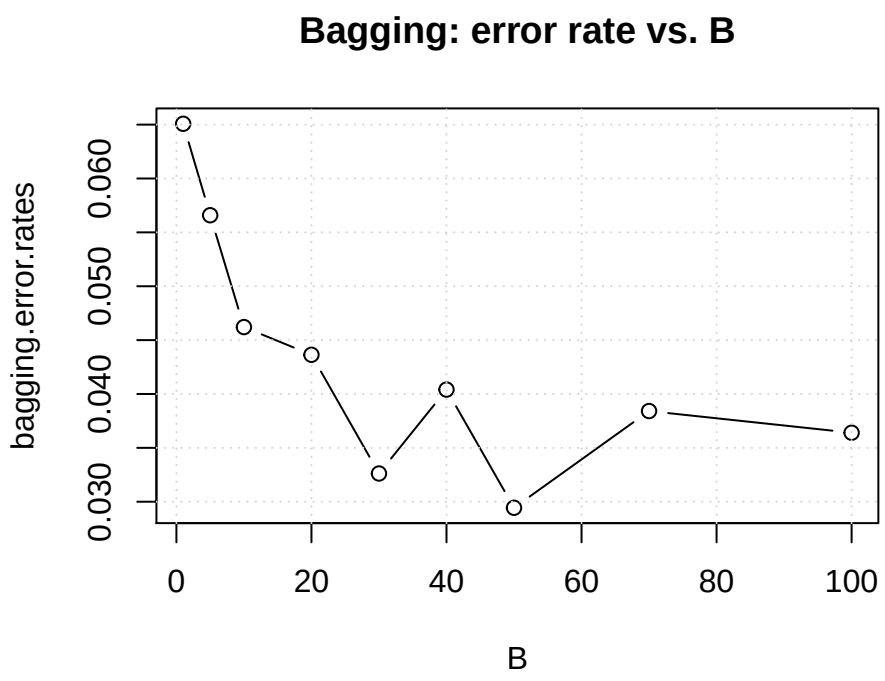
```
## [1] 0.2833333
```

Obserwacje:

- Cały zbiór: Błąd na poziomie mniej niż 1% dla zbioru uczącego do około 7% dla testowego, co zgadza się z wynikami z poprzedniego raportu.
- Zmienne o najlepszej zdolności dyskryminacyjnej: Błąd na poziomie mniej niż 16% dla zbioru uczącego do około 28% dla testowego, co jest nieco gorszym rezultatem względem poprzedniego raportu (tam 14-22%).

1.2.2 Algorytm *bagging*

Zaczniemy od wykresu zmiany błędu klasyfikacji względem liczby replikacji. Użyjemy całego zbioru danych:



Rysunek 1: Bagging. Zależność dokładności od liczby replikacji B

Optymalna liczba replikacji wynosi 50, (najniższa wartość na wykresie) takiej też użyjemy w naszych rozważaniach.

Cały zbiór danych

- Grupa ucząca

  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	35	0	0
2	0	47	0
3	0	0	36

Tabela 5: Macierz pomyłek - bagging, grupa ucząca

  Błąd klasyfikacyjny:

```
## [1] 0
```

- Grupa testowa

  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	24	0	0
2	1	22	1
3	0	0	12

Tabela 6: Macierz pomyłek - bagging, grupa testowa

Błąd klasyfikacyjny:

```
## [1] 0.03333333
```

Zmienne o najlepszej zdolności dyskryminacyjnej

- Grupa ucząca

  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	35	0	0
2	0	47	0
3	0	0	36

Tabela 7: Macierz pomyłek - bagging, grupa ucząca

Błąd klasyfikacyjny:

```
## [1] 0
```

- Grupa testowa

  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	17	7	0
2	7	14	3
3	0	2	10

Tabela 8: Macierz pomyłek - bagging, grupa testowa

Błąd klasyfikacyjny:

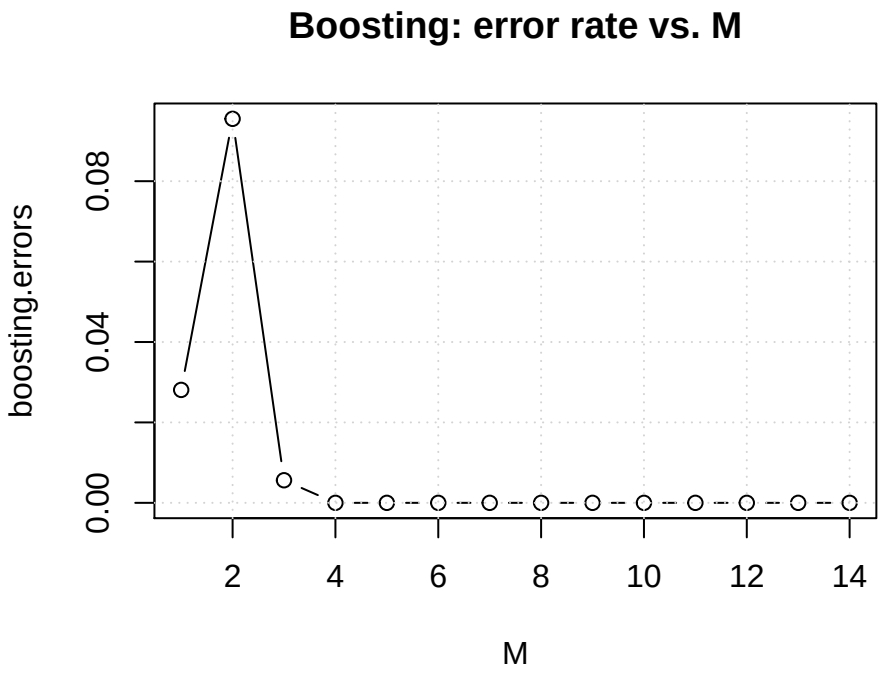
```
## [1] 0.3166667
```

Obserwacje:

- Cały zbiór: Błąd na poziomie 0 % dla zbioru uczącego do około 3% dla testowego.
- Zmienne o najlepszej zdolności dyskryminacyjnej: Błąd na poziomie 0% dla zbioru uczącego do około 32% dla testowego.

1.2.3 Algorytm AdaBoost

Zaczniemy od przedstawienia tego, jak zmienia się błąd klasyfikacji w zależności od liczby wykonywanych w algorytmie iteracji. Zmieniamy metodę Z 632+ na "predict boosting" dla całego zbioru danych celem optymalizacji czasu kompilacji.



Rysunek 2: AdaBoost. Zależność dokładności od liczby iteracji M

Od M=4 mamy stałą wartość błędu bliską 0, stąd dalszą analizę przeprowadzimy dla M=4 iteracji. Wyniki porównamy z M=10 iteracjami.

Cały zbiór danych

- Grupa ucząca

  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	35	0	0
2	0	47	0
3	0	0	36

Tabela 9: Macierz pomyłek - boosting, grupa ucząca

  Błąd klasyfikacyjny:

```
## [1] 0
```

- Grupa testowa

Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	24	0	0
2	0	24	0
3	0	0	12

Tabela 10: Macierz pomyłek - boosting, grupa testowa

Błąd klasyfikacyjny:

```
## [1] 0
```

Zmienne o najlepszej zdolności dyskryminacyjnej

- Grupa ucząca

  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	34	1	0
2	2	40	5
3	0	2	34

Tabela 11: Macierz pomyłek - boosting, grupa ucząca

Błąd klasyfikacyjny:

```
## [1] 0.08474576
```

- Grupa testowa

  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	18	6	0
2	7	15	2
3	0	2	10

Tabela 12: Macierz pomyłek - boosting, grupa testowa

Błąd klasyfikacyjny:

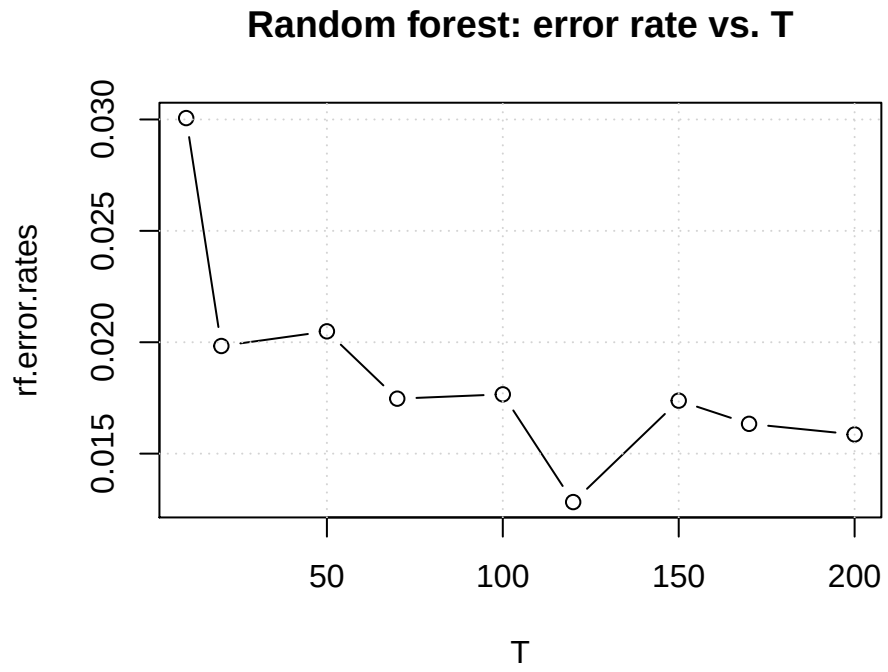
```
## [1] 0.2833333
```

Obserwacje:

- Cały zbiór: Błąd na poziomie mniej niż 2% dla zbioru uczącego do 5% dla testowego. Po zmianie na M = 10 błąd spada do 0 dla obu zbiorów.
- Zmienne o najlepszej zdolności dyskryminacyjnej: Błąd na poziomie mniej niż 14% dla zbioru uczącego do około 20% dla testowego. Po zmianie na M=10 wartość błędu zmienia się na odpowiednio 8% i 28%.

1.2.4 Lasy losowe

Klasycznie dla tego zadania rozpoczynamy od wykresu zależności dokładności od liczby drzew dla całego zbioru.



Rysunek 3: Random forest. Zależność dokładności od liczby drzew T

Wzrost liczby drzew nie gwarantuje minimalizacji błędu, optymalna wartość T =120 drzew.

Cały zbiór danych

- Grupa ucząca
  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	35	0	0
2	1	44	2
3	0	0	36

Tabela 13: Macierz pomyłek - random forest, grupa ucząca

  Błąd klasyfikacyjny:

```
## [1] 0.02542373
```

- Grupa testowa
  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	24	0	0
2	0	23	1
3	0	0	12

Tabela 14: Macierz pomyłek - random forest, grupa testowa

Błąd klasyfikacyjny:

```
## [1] 0.01666667
```

Zmienne o najlepszej zdolności dyskryminacyjnej

- Grupa ucząca

  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	31	4	0
2	12	28	7
3	0	5	31

Tabela 15: Macierz pomyłek - random forest, grupa ucząca

  Błąd klasyfikacyjny:

```
## [1] 0.2372881
```

- Grupa testowa

  Macierz błędu:

	1 - przewidywane	2 - przewidywane	3 - przewidywane
1	19	5	0
2	6	16	2
3	0	2	10

Tabela 16: Macierz pomyłek - random forest, grupa testowa

  Błąd klasyfikacyjny:

```
## [1] 0.25
```

Obserwacje:

- Cały zbiór: Błąd na poziomie 2% dla obu zbiorów. Po zmianie na T = 60 błąd ptaktycznie się nie zmienia.
- Zmienne o najlepszej zdolności dyskryminacyjnej: Błąd na poziomie 24-25% dla obu grup. Po zmianie na T=60 rezultaty są niemal identyczne.
- Wyniki nie zmieniają się mocno względem zbioru uczącego i testowego, z czym nie mieliśmy wcześniej do czynienia. Oprócz tego potwierdziliśmy, że zwiększenie ilości drzew T nie zmienia bardzo dokładności.

1.2.5 Porównanie metod

Wyliczamy błąd klasyfikacyjny za pomocą metody632+

```
p <-  ncol(mydata) - 1
error.tree         <- (errorest(Type~., data=mydata, model=rpart, predict=mypredict.rpart,estimator="632pl
error.tree$error #drzewo

## [1] 0.09256317

error.bagging      <- (errorest(Type~., data=mydata, model=bagging, nbagg=50, minsplit=1,cp=0,estimator="6
error.bagging$error #bagging

## [1] 0.03382883

error.randomForest <- (errorest(Type~., data=mydata, model=randomForest,ntree=120,mtry=floor(sqrt(p)),esti
error.randomForest$error #las

## [1] 0.0118525

error.boost <- (errorest(Type~., data=mydata, model=boosting, predict=mypredict.boosting,mfinal=10,estimat
error.boost$error #ada

## [1] 0.02506827

error.tree1         <- (errorest(Type~., data=mydata[,c(1,7,8)], model=rpart, predict=mypredict.rpart,esti
error.tree1$error #drzewo podzbiór

## [1] 0.2143104

error.bagging1      <- (errorest(Type~., data=mydata[,c(1,7,8)], model=bagging, nbagg=50, minsplit=1,cp=0,
error.bagging1$error #bagging podzbiór

## [1] 0.1919434

error.randomForest1 <- (errorest(Type~., data=mydata[,c(1,7,8)], model=randomForest,ntree=120,mtry=floor(s
error.randomForest1$error #las podzbiór

## [1] 0.2019268

error.boost1 <- (errorest(Type~., data=mydata[,c(1,7,8)], model=boosting, predict=mypredict.boosting,mfinal
error.boost1$error #ada podzbiór

## [1] 0.2113367
```

Wyliczamy redukcję błędu całego zbioru danych:

```
tree.bagg<-(error.tree$error - error.bagging$error)/error.tree$error*100
tree.bagg

## [1] 63.45325

tree.rf<-(error.tree$error - error.randomForest$error)/error.tree$error*100
tree.rf

## [1] 87.19523

tree.boost<-(error.tree$error - error.boost$error)/error.tree$error*100
tree.boost

## [1] 72.91766
```

```
#Pozdbiór:
t.bagg<-(tree.blad.t - bagging.blad.t)/tree.blad.t*100
t.bagg

## [1] 50

t.boost<-(tree.blad.t - boosting.blad.t)/tree.blad.t*100
t.boost

## [1] 100

t.rf<-(tree.blad.t - rf.blad.t)/tree.blad.t*100
t.rf

## [1] 75
```

Wyliczamy redukcję błędu dla pozdbioru danych o najlpeszej zdolności dyskryminacyjnej:

```
tree.bagg1<-(error.tree1$error - error.bagging1$error)/error.tree1$error*100
tree.bagg1

## [1] 10.43671

tree.rf1<-(error.tree1$error - error.randomForest1$error)/error.tree1$error*100
tree.rf1

## [1] 5.778364

tree.boost1<-(error.tree1$error - error.boost1$error)/error.tree1$error*100
tree.boost1

## [1] 1.387553

#Pozdzbiór:
t.bagg1<-(tree.blad.t1 - bagging.blad.t1)/tree.blad.t1*100
t.bagg1

## [1] -11.76471

t.boost1<-(tree.blad.t1 - boosting.blad.t1)/tree.blad.t1*100
t.boost1

## [1] 0

t.rf1<-(tree.blad.t1 - rf.blad.t1)/tree.blad.t1*100
t.rf1

## [1] 11.76471
```

Wyniki otrzymane metodą macierzy pomyłek dla zbioru testowego wszystkich danych oraz zmiennych o najlepszej zdolności dyskryminacyjnej porównamy z otrzymanymi metodą 632+ dla całego zbioru. Wyznaczymy względną redukcję błędu dla każdego algorytmu.

- BAGGING vs DRZEWO
  -CAŁY ZBIÓR: Błąd klasyfikacyjny: 3,3% do 6,7% w przypadku macierzy pomyłek, co daje redukcję błędu na poziomie 50%, 3,4% do 9,3% w przypadku metody 632+, co daje 63,5% redukcji błędu.

-ZMIENNE O NAJLEPSZEJ ZDOLNOŚCI DYSKRYMINACYJNEJ: 31,7% do 28,3% w przypadku macierzy pomyłek (brak redukcji błędu), 19,2% do 21,4% metodą 632+, co daję redukcję błędu na poziomie 10%

- BOOSTING vs DRZEWO
  -CAŁY ZBIÓR: Błąd klasyfikacyjny: 0% do 6,7% w przypadku macierzy pomyłek, co daje redukcję błędu na poziomie 100%, 2,5% do 9,3% w przypadku metody 632+, co daje 73% redukcji błędu.

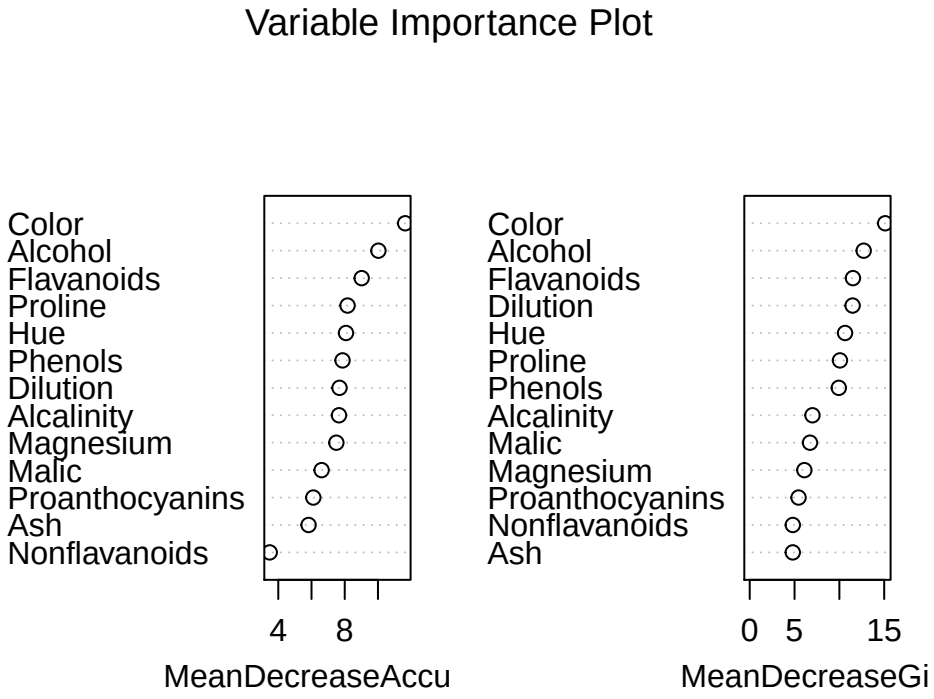
  -ZMIENNE O NAJLEPSZEJ ZDOLNOŚCI DYSKRYMINACYJNEJ: 28,3% do 28,3% w przypadku macierzy pomyłek (brak redukcji błędu), 21,1% do 21,4% metodą 632+, co daję redukcję błędu na poziomie 1,4%

- LASY LOSOWE vs DRZEWO
  -CAŁY ZBIÓR: Błąd klasyfikacyjny: 1,7% do 6,7% w przypadku macierzy pomyłek, co daje redukcję błędu na poziomie 75%, 1,2% do 9,3% w przypadku metody 632+, co daje 87% redukcji błędu.

  -ZMIENNE O NAJLEPSZEJ ZDOLNOŚCI DYSKRYMINACYJNEJ: 25% do 28,3% w przypadku macierzy pomyłek (redukcja na poziomie 11,8%), 20,2% do 21,4% metodą 632+, co daję redukcję błędu na poziomie 5,8%

1.2.6 Wybór najistotniejszych zmiennych

Które zmienne są najistotniejsze według algorytmu random forest?



Rysunek 4: Ranking cech według metody lasów losowych

Proline i Flavanoids przodują w rankingu. Pierwsza z nich być może przez dużą wariancję, druga była w gronie zmiennych o najlepszej zdolności dyskryminacyjnej. Dalej mamy Hue i Alcohol. Co ciekawe, zmienna

Phenols miała drugą najlepszą zdolność dyskryminacyjną, jednak w tym zestawieniu uplasowała się dokładnie pośrodku. Najgorsza wydaje się być zmienna Nonflavanoids.

1.2.7 Podsumowanie

- Algorytmem zapewniającym największą redukcję błędu całego zbioru danych jest AdaBoost, jednak w przypadku zmiennych o wysokiej zdolności dyskryminacyjnej nie jest już tak efektowny (raptem 1,4%). Jeśli dodamy do tego ogromnie długi czas pracy (najdłuższy ze wszystkich algorytmów), okaże się, że lepszym wyborem są lasy losowe.
- Lasy losowe nieznacznie gorzej prezentują się gdy analizujemy cały zbiór danych, jednak dają więcej gdy analizujemy podzbiór danych o wysokiej zdolności dyskryminacyjnej. Dodając do tego średni czas pracy, uzyskujemy najlepszy algorytm z naszego zestawienia.
- Najsłabszy jest algorytm bagging (ponad 50% różnicy między drugim AdaBoostem). Jego jedyną zaletą jest to, że jest najszybszy w naszym zestawieniu

1.3 Metoda wektorów nośnych - SVM

U podstaw metody wektorów nośnych (Support Vector Machines - SVM) leży koncepcja przestrzeni decyzyjnej, którą dzieli się budując granice separujące obiekty o różnej przynależności klasowej.

1.3.1 SVM - jądro liniowe i różne wartości parametru kosztu C

Rożważmy jądro liniowe, a więc problem doboru optymalnej wartości kosztu C, która określa szerokość pasma oraz wpływa na dokładność klasyfikacji. W naszych danych z pakietu *HDclassif* o nazwie "WINE" wiemy, że zmienną o zdolnościach klasyfikacyjnych jest cecha TYPE. Stworzymy dwa modele SVM, jeden oparty na całym zbiorze danych oraz drugi na wybieranych trzech zmienne(Proline, Colors, Flavanoids), które we wcześniejszym raporcie uznaliśmy za jedne z najlepszych do podziału na tym zbiorze.

Na stworzonych modelach z jądrami liniowymi porównany wykresy od zmiennych tworzących te modele (Proline i Colors) oraz zbadamy róznice, jak kształtuje się podziału na klasy i ich przyporządkowania, gdy wykres narysuje od zmiennych Proline i Hue, gdzie zmienna Hue nie tworzyła modelu drugiego.

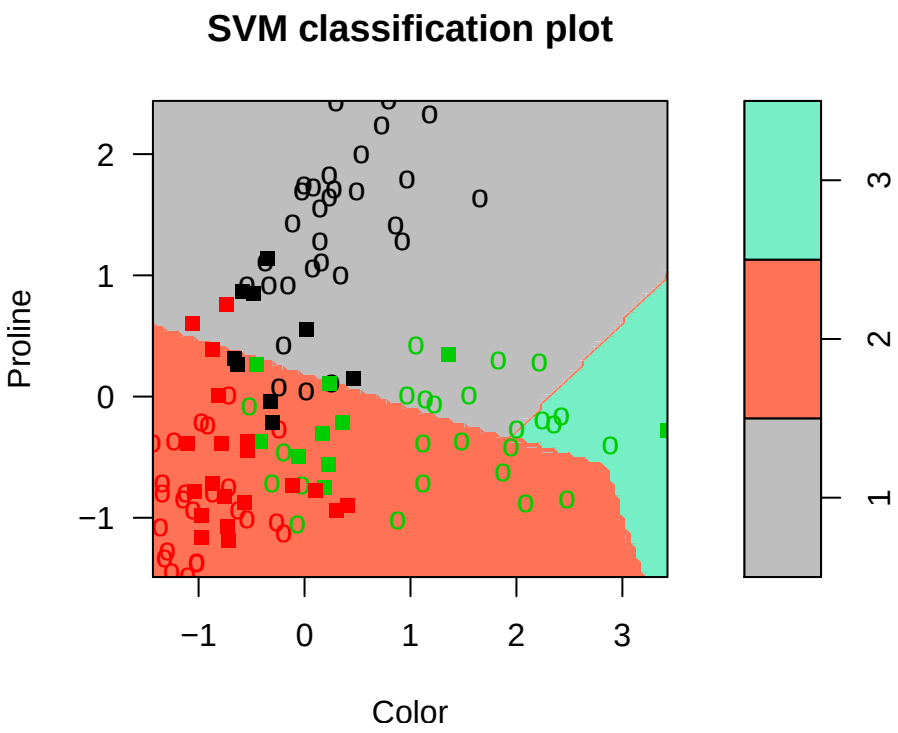
- Parametr kosztu: $C = 0.1$. Tworzymy odpowiednie modele:

```
#1.  cały zbiór
svm.linear.C0.1.all <- svm(Type~., data=training.set, kernel="linear", cost=.1)
#2.  zmienne Proline, Color, Flavanoids
svm.linear.C0.1.sub <- svm(Type~Proline+Color+Flavanoids, data=training.set, kernel="linear", cos
```

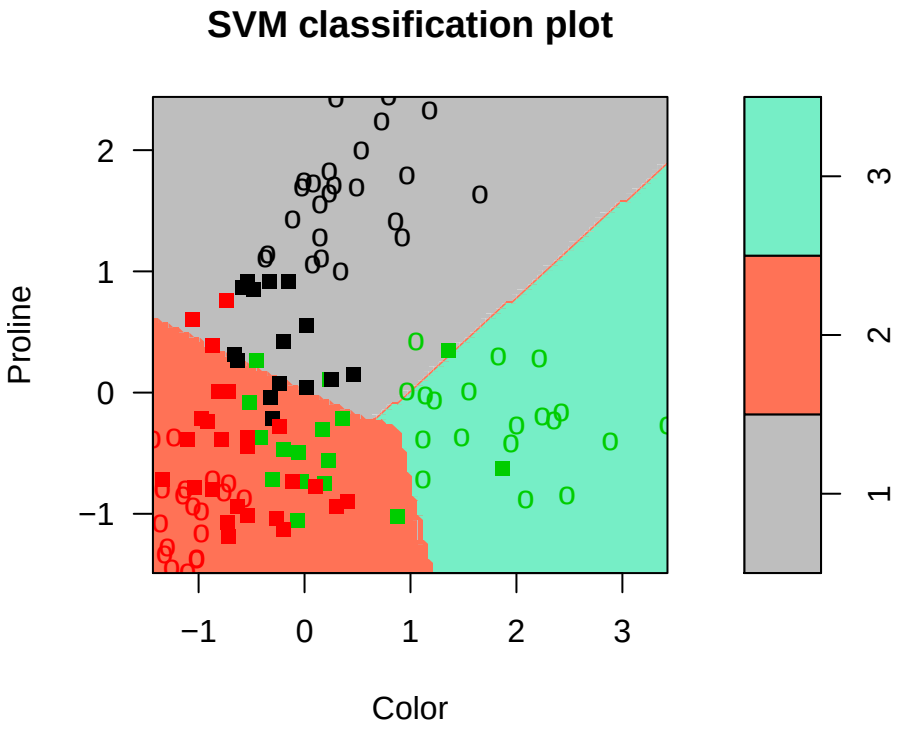
  Poniżej porównanie podziału na zbiorze uczącym do właściwych Typów Win, gdzie:

  * Czarny - Typ 1
  * Czerowny - Typ 2
  * Zielony - Typ 3

  Wykresy dwóch modelów SVM dla $C = 0.1$ dla zmiennych *Proline* i *Color*.

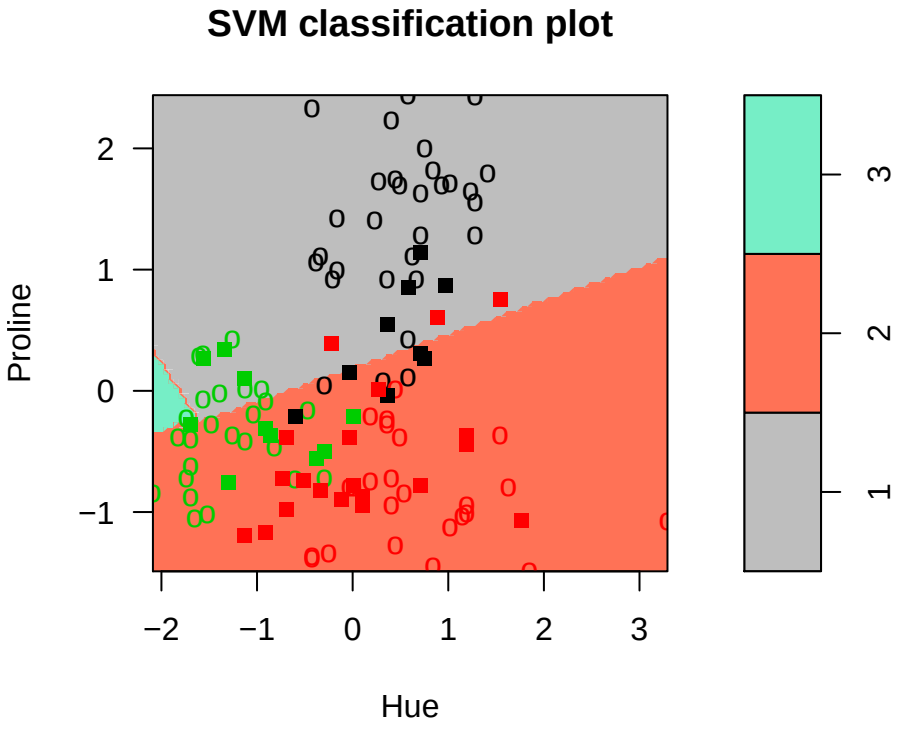


Rysunek 5: C=0.1, Zmienne występujące

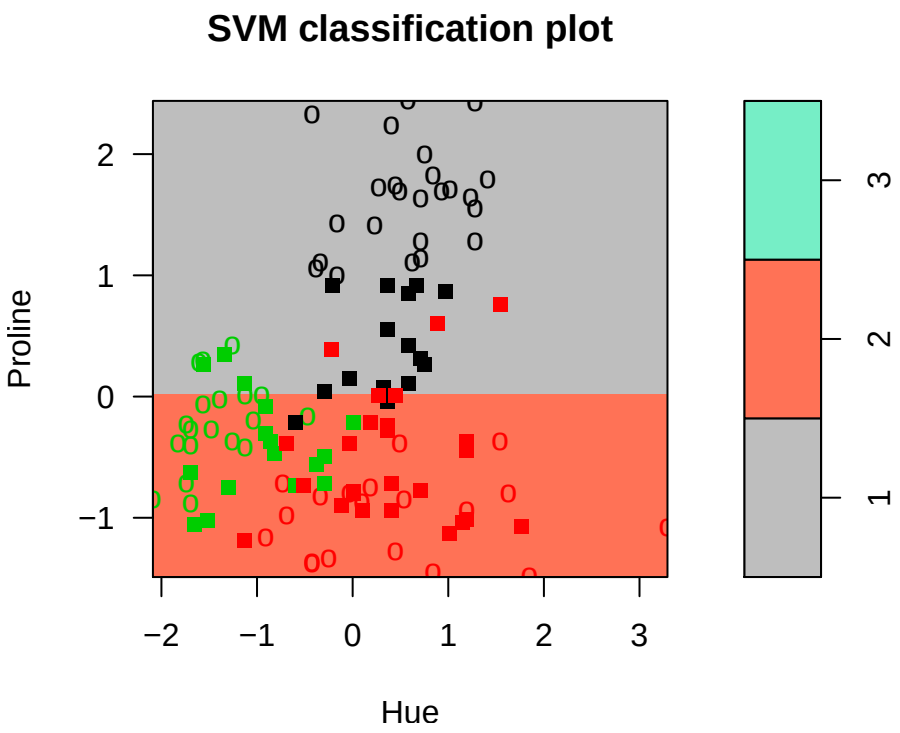


Rysunek 6: C=0.1, Zmienne występujące

Oraz dla tej samej wartości C, ale dla zmiennych *Proline* i *Hue*

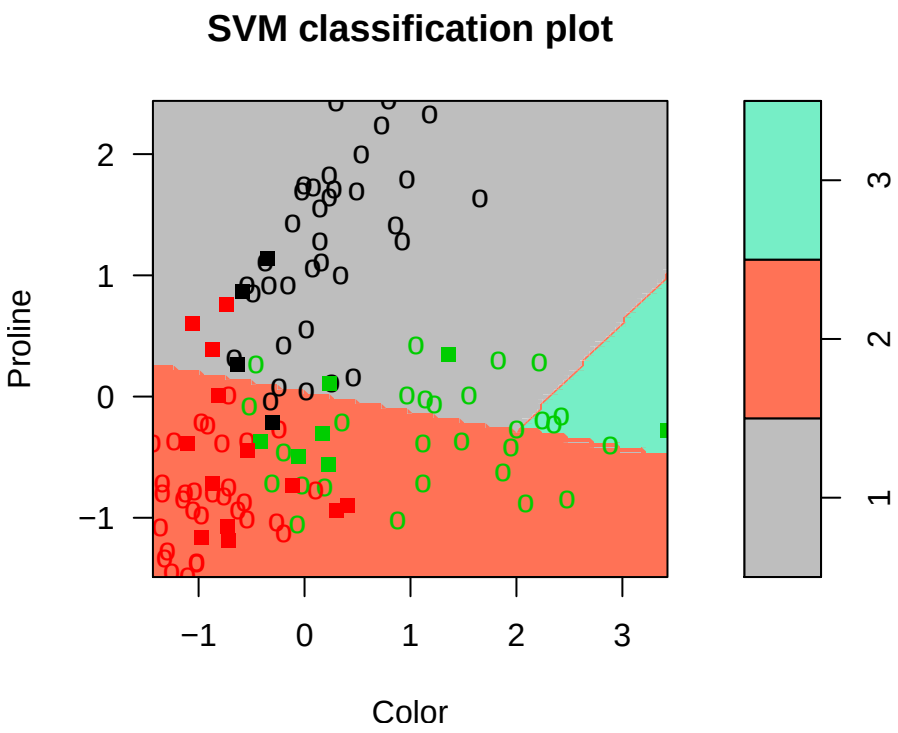


Rysunek 7: C=0.1, Jedna zmienna niewystępująca

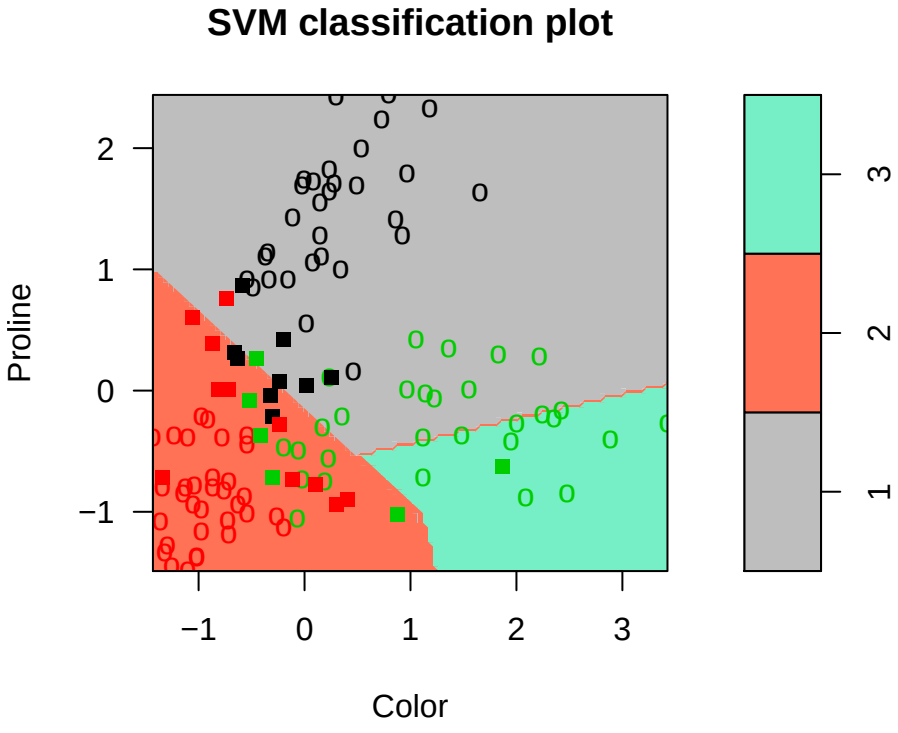


Rysunek 8: C=0.1, Jedna zmienna niewystępująca

– Parametr kosztu: $C = 1$ Tworzymy odpowiednie modele:
Wykresy dwóch modelów SVM dla $C = 1$ dla Zmiennych *Proline* i *Color*

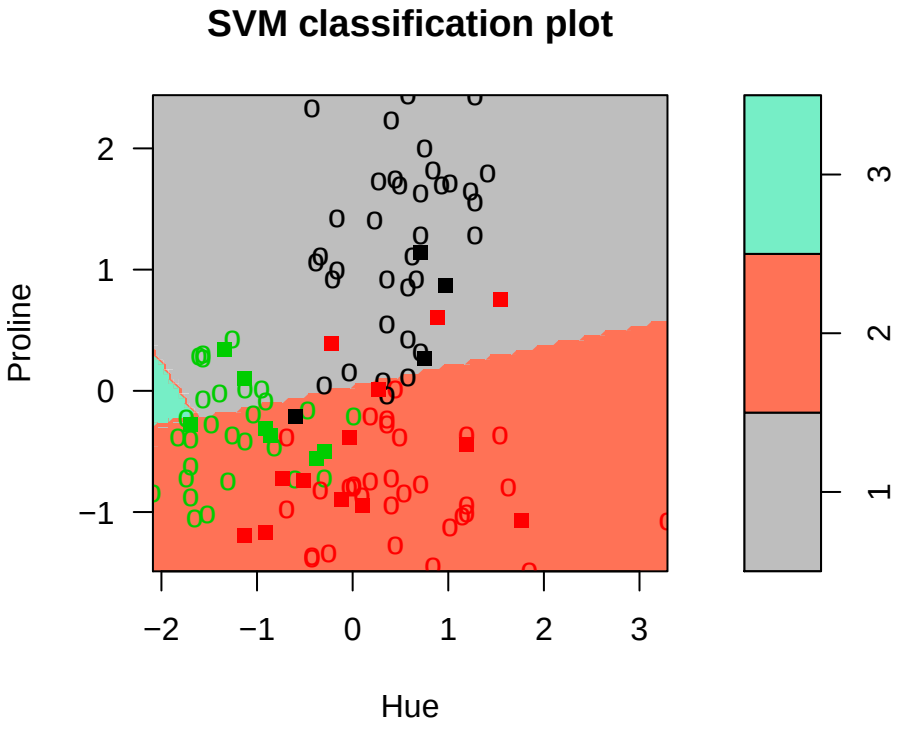


Rysunek 9: C=1, Zmienne występujące

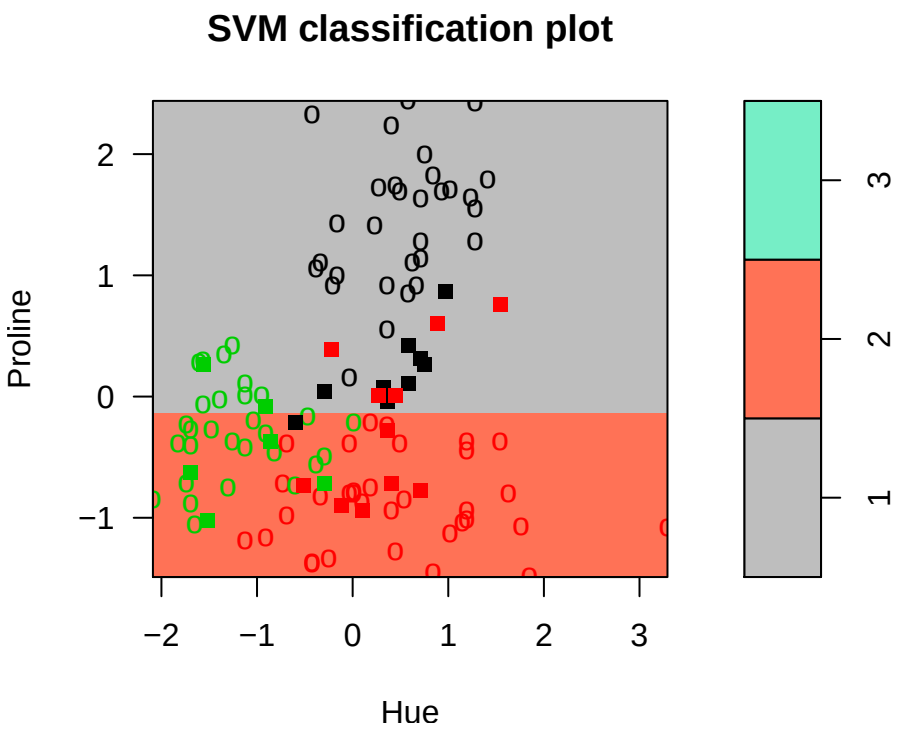


Rysunek 10: C=1, Zmienne występujące

Oraz dla tej samej wartości C, ale dla zmiennych *Proline* i *Hue*

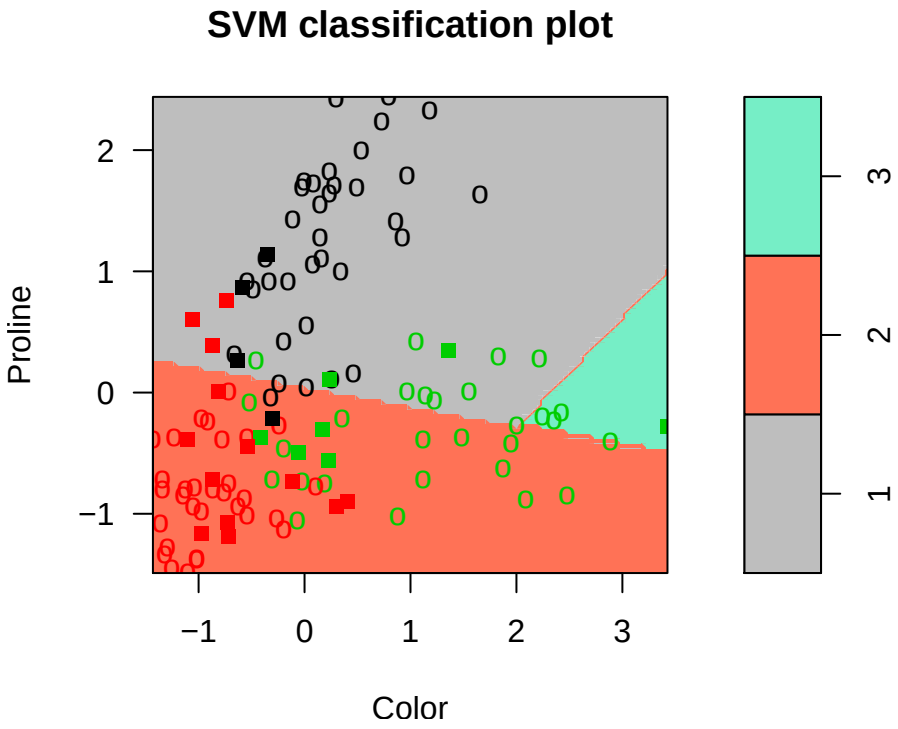


Rysunek 11: C=1, Jedna zmienna niewystępująca

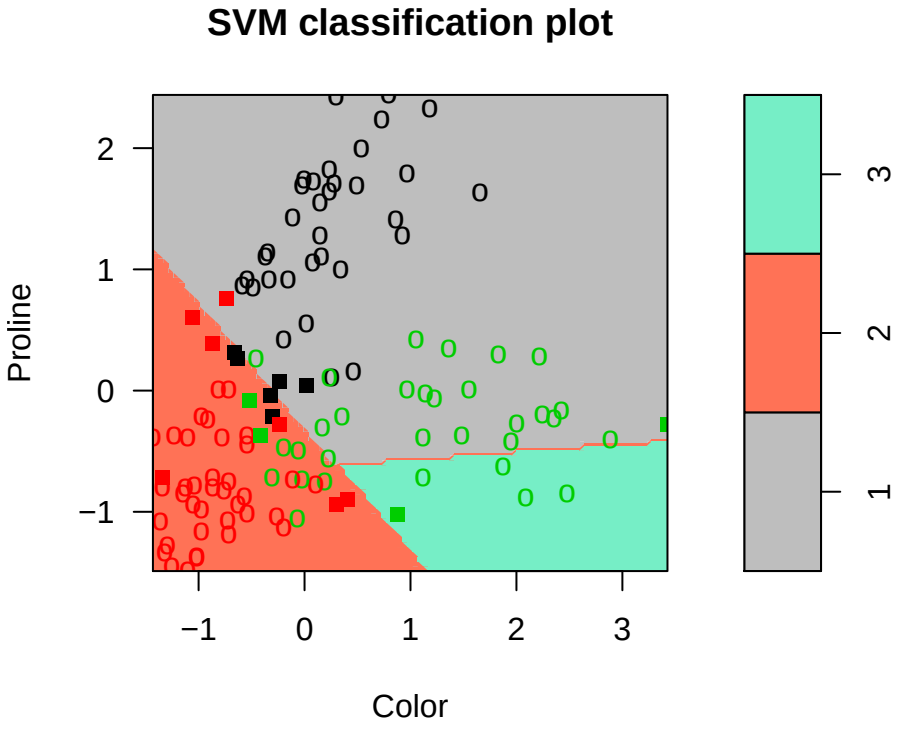


Rysunek 12: C=1, Jedna zmienna niewystępująca

– Parametr kosztu $C = 20$ Tworzymy odpowiednie modele:
Wykresy dwóch modelów SVM dla $C = 20$ dla Zmiennych *Proline* i *Color*

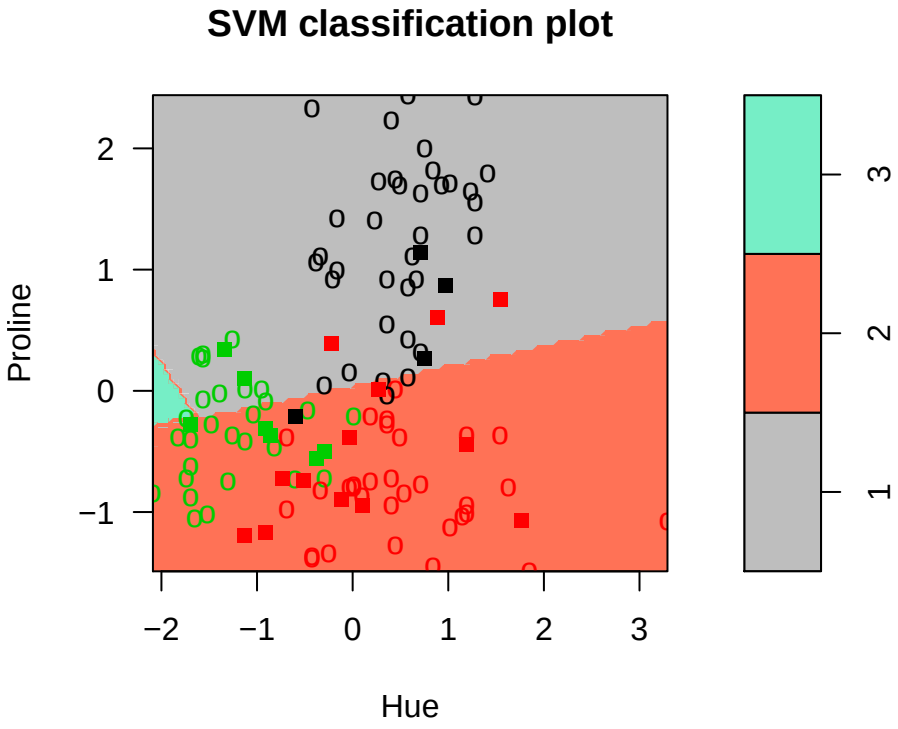


Rysunek 13: C=20, Zmienne występujące

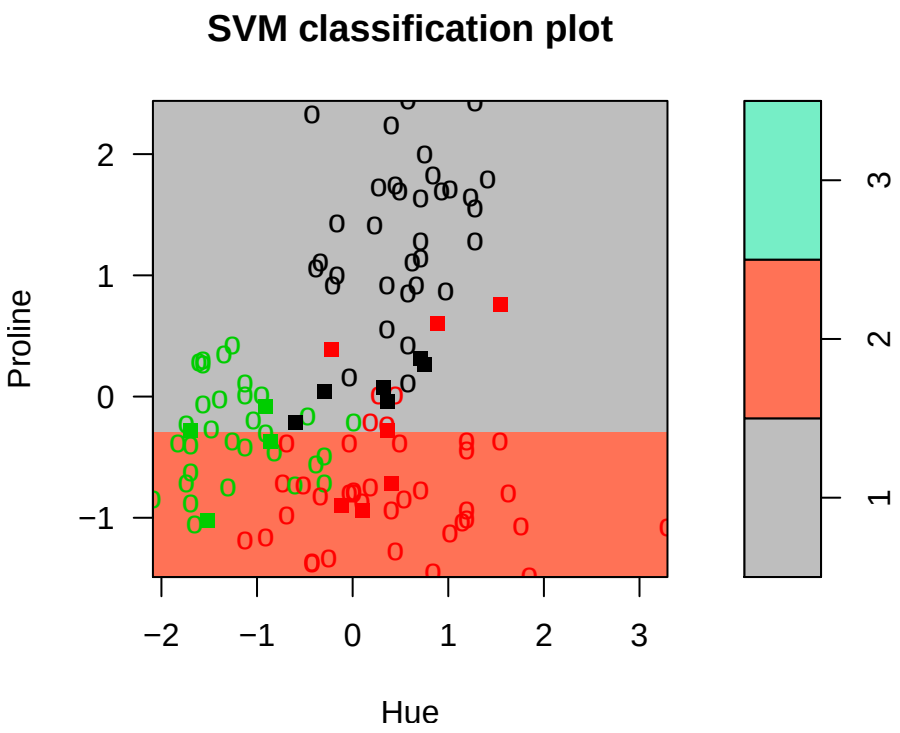


Rysunek 14: C=20, Zmienne występujące

Oraz dla tej samej wartości C, ale dla zmiennych *Proline* i *Hue*



Rysunek 15: C=20, Jedna zmienna niewystępująca



Rysunek 16: C=20, Jedna zmienna niewystępująca

Wnioski po wykresach:

- W obu przypadkach mocno mieszają się Wina o Typach 2 oraz 3(czerowny i zielony),
- Najgorzej dopasowany parametr kosztu w obu przypadkach występuje dla wartośći C = 20
- Podział klasyfikacyjny na 3 klasy jest prawie niewidoczny na wykresie zależnośći Proline i Hue, gorzej jednak wygląda to w przypadku modelu 2
- Natomiast na wykresie zależności Proline i Colors lepszym modelem wydaje się ten skonstruowany na 3 zmiennych, a najlepszym parametrem C jest 0.1

Porównajmy, czy nasze obserwacje potwierdzają się w obliczeniach. Rozważmy zbiór testowy dla obu modeli, róznych zmiennych C i sprawdźmy prognozy i ich dokładności

- Parametr kosztu: C = 0.1

```
## [1] "Model 1 --- C=0.1"
## [1] 1
## [1] "Model 2 --- C=0.1"
## [1] 0.9833333
```

- Parametr kosztu C = 1

```
## [1] "Model 1 --- C=1"
## [1] 0.9833333
## [1] "Model 2 --- C=1"
## [1] 1
```

– Parametr kosztu C = 20

```
## [1] "Model 1 --- C=20"
## [1] 0.9833333
## [1] "Model 2 --- C=20"
## [1] 0.9666667
```

Test innego modelu zbudowanego na zmiennej Flavanoids i Nonflavanoids - najlepsze zdolności dyskryminacyjne

```
## [1] "Model 3 --- C=0.1"
## [1] 0.7333333
## [1] "Model 3 --- C=1"
## [1] 0.7666667
## [1] "Model 3 --- C=20"
## [1] 0.75
```

Wnioski: - Najlepszym parametrem dla tego jądra okazała się wartośc C = 1 w przypadku trzech modeli - Wahania niedopasowań dla różnych modeli są na poziomie $2 - 4\%$ - Widzimy, że otrzymujemy bardzo dobre przewidywanie już na modelu zbudowanym przez 3 cechy - Dla całego zbioru najlepszy parametr kosztu jest na poziomie 0.1, natomiast dla modelu 2 w okolicach 1

1.3.2 SVM - badanie różnych jąder do budowania klasyfikatorów

Będziemy rozważać jądro liniowe z parametrem C = 1, wielomianowe(wykrzywia prostą) oraz radialne dla 3 modeli zaproponowanych wcześniej.
Model 1:

```
## [1] "Model 1 --- C=1, jądro liniowe"
## [1] 0.9833333
## [1] "Model 1 --- st.wielomianu=1, jądro wielomianowe"
## [1] 1
## [1] "Model 1 --- st.wielomianu=5, jądro wielomianowe"
## [1] 0.8666667
## [1] "Model 1 --- gamma=0.1, jądro radialne"
## [1] 1
## [1] "Model 1 --- gamma=1, jądro radialne"
## [1] 0.6833333
```

Model 2:

```
## [1] "Model 2 --- C=1, jądro liniowe"
## [1] 1
## [1] "Model 2 --- st.wielomianu=1, jądro wielomianowe"
## [1] 0.9833333
## [1] "Model 2 --- st.wielomianu=5, jądro wielomianowe"
## [1] 0.7333333
## [1] "Model 2 --- gamma=0.1, jądro radialne"
## [1] 0.9833333
## [1] "Model 2 --- gamma=1, jądro radialne"
## [1] 0.95
```

Model 3:

```
## [1] "Model 3 --- C=1, jądro liniowe"
## [1] 0.7666667
## [1] "Model 3 --- st.wielomianu=1, jądro wielomianowe"
## [1] 0.7666667
## [1] "Model 3 --- st.wielomianu=5, jądro wielomianowe"
## [1] 0.6666667
## [1] "Model 3 --- gamma=0.1, jądro radialne"
## [1] 0.75
## [1] "Model 3 --- gamma=1, jądro radialne"
## [1] 0.7666667
```

Wnioski:

- Dla wielu modeli najlepiej spisuje się jądro radialne z parametrem gamma = 0.1
- Jądro uależnione od stopnia wielomianu jest abrdzo wrażliwe na jego stopień
- O dziwo jądro liniowe spisuje się bardzo dobrze, więc możemy uznać że nasze dane dzielą się dobrze klasowo, iż można je podzielić liną prostą i stworzymy w miarę dobry model podziału

1.3.3 SVM - optymalizacja parametrów gamma i C w jądrze radialnym

Rozważmy tylko model 3, w którym możemy zobaczyć największą różnicę w wyborze tych parametrów

```
## [1] "Model 3 --- najlepsza C = 4, najlepsza gamma = 1"
## [1] "Najlepsza metoda przewidywań:"
## [1] 0.75
```

Wniosek: Jako, że rozważaliśmy gamma = 1 dla tego modelu zauważamy, że zmienna C nie ma dużego wpływu w tym modelu podobnie było gdy rozważaliśmy tylko dla jądra liniowego

2 Zadanie 2 - Analiza skupień

W tym zadaniu rozważymy działanie algorytmów na bazie analizy skupień, gdzie porównamy jakość grupowania wybranych metod (algorytm grupujący PAM i hierarchiczny AGNES) i wybierzemy optymalne parametry dla danych metod.

2.1 Analiza skupień - algorytm grupujący (PAM)

Algorytm Grupujący PAM, inaczej K-medoidów wykorzystuje inne miary odległości między obiektami niż k-średnich, co wpływa na lepsze poszukiwanie zależności w modelach z większą róznicą między odległościami, jendka za cenę wyższej złożoności obliczeniowej. Metoda działania algorytmu PAM:

- Zainicjujemy algorytm wybierając losowe obiekty jako medoidy/centra/reprezentantów grup.
- Dla wszystkich obiektów wyznaczymy ich przypisanie na zasadzie odległości od najbliższego medoidu.
- Dla każdej grupy, sprawdzamy czy inny obiekt z tej grupy nie ma mniejszej sumy odległości od wszystkich pozostałych w tej grupie. Jeżeli tak, to to on powinien być nowym medoidem.
- Powtarzamy kroki 2-3 tak długo póki zmienia się przypisanie do grup.

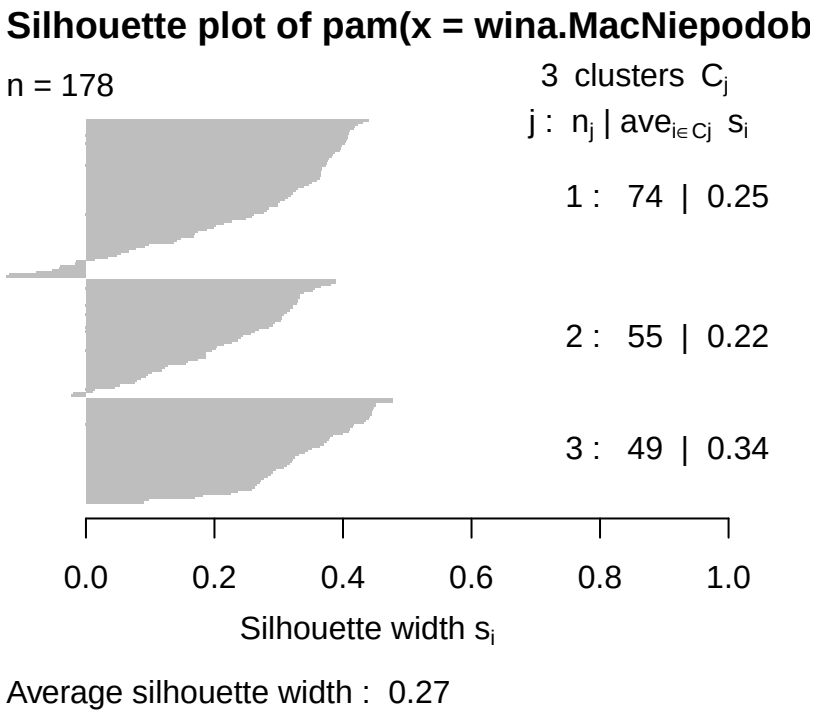
2.1.1 Wizualizacja wyników grupowania

Wprowadzamy zmienne i standaryzujemy je:

```
library(HDclassif)
data(wine)
colnames(wine) <- c('Type', 'Alcohol', 'Malic', 'Ash',
                    'Alcalinity', 'Magnesium', 'Phenols',
                    'Flavanoids', 'Nonflavanoids',
                    'Proanthocyanins', 'Color', 'Hue',
                    'Dilution', 'Proline')
wine$Type <- as.factor(wine$Type)
etykietki<-wine$Type
etykietki<-data.frame(etykietki)
wine1<-wine[,-1]
wine1<-scale(wine1)
wine1<-data.frame(wine1)
```

Tworzymy nasz model PAM dla liczby klas równej 3, gdyż mamy znienną klasyfikującą TYPE określającą trzy odmiany wina, zobaczymy czy istnieje taki podział stowrozny przez algorytm grupujący i jak ma się on do realnych klas, następnie zaczniemy testować, czy może istnieją inne skupienia naszego zbioru:

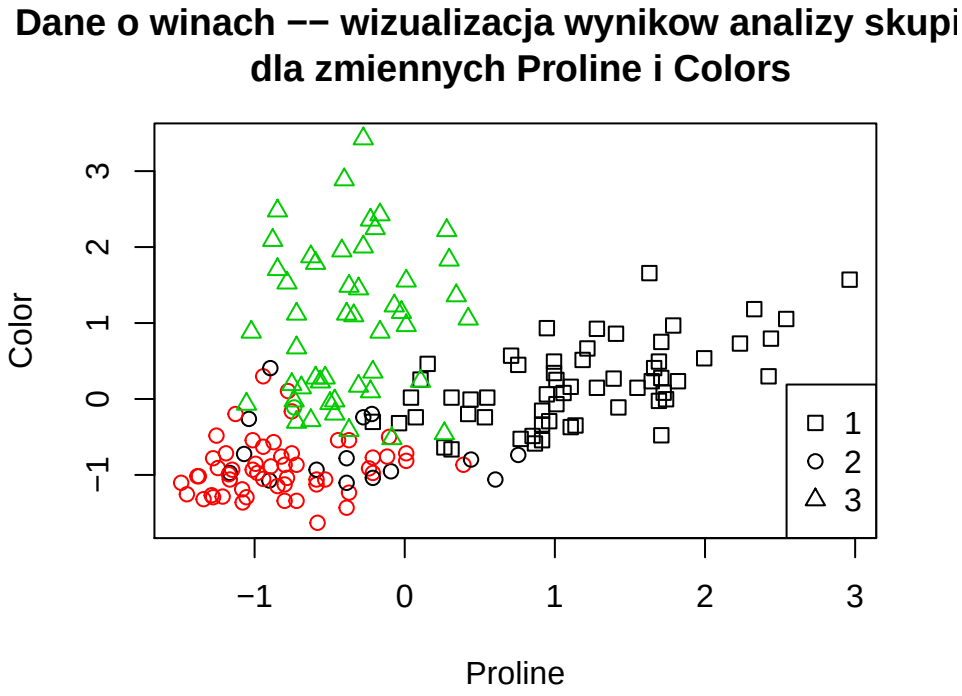
```
## [1] "Wartości odpowiednich przyporządkowań i ich dopasowanie (-1,1) do klastera"
```



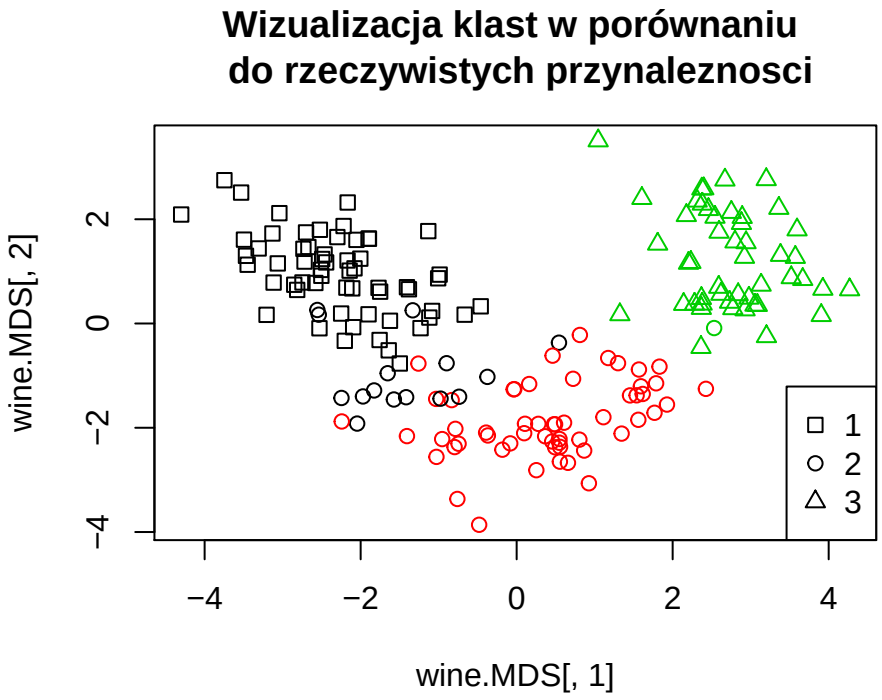
Widzimy, że przy wyborze klasy cluster 2 istnieje mieszanie się sąsiadów z grupy 1. Na wysokim poziomie $+/-$ 0.4 są odległości podobieństwa w klasie 3(analiza na podstawie funckji wine.pam3$silinfo$), $najgorzejwtymwypadaklasa2.Sp$

2.1.2 Interpretacja wyników i ocena dokładności

Zobaczmy jak wygląda podział na odpowiednie klasy(clustery) w wykresie zależności *Proline* i *Color*



Popatrzmy jak można to lepiej przedstawić, skonstruujemy model oparty na metodzie MDS(moglibyśmy PCA, bo mamy same liczbowe) i zobaczymy jak wygląda rzeczywisty podział do algorytmu grupującego za pomocą wykresu rozproszenia



Rysunek 17: Wizualizacja K=3

Sprawdźmy jak wygląda nasz model. Nasze medoidy oraz porównanie ile elementów dobrze trafiło do swojej klasy.

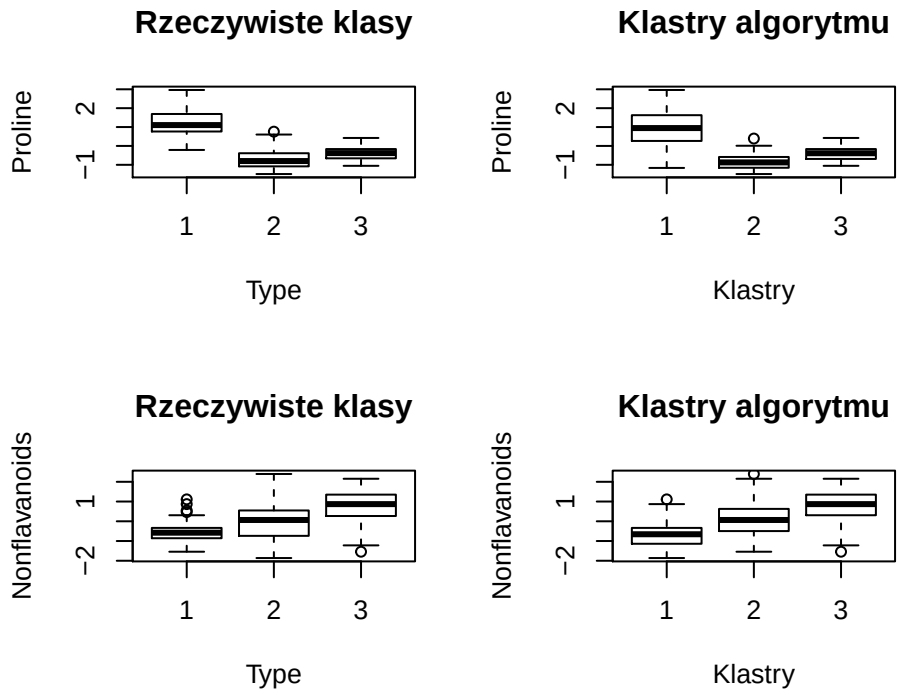
```
## [1] "Medoidy dla K=3 (odpowiednie wiersze):"
## [1] "36"  "107" "149"
## wine.pam3$clustering: 1
##
##  1  2  3
## 59 15  0
## ------------------------------------------------------------
## wine.pam3$clustering: 2
##
##  1  2  3
##  0 55  0
## ------------------------------------------------------------
## wine.pam3$clustering: 3
##
##  1  2  3
##  0  1 48
```

Rzeczywiście klasa druga miesza się z pierwszą przez co typ wina o numerze 1 wydaje się najliczniejszy, a w naszych danych to wina o typie 2 jest najwięcej. Sprawdźmy w ilu procentach nasze zmienne się zgadzają co do rzeczywistych typów:

```
##                etykietki.rzeczywiste
## etykietki.pam3  1  2  3
##              1 59 15  0
```

```
##              2  0 55  0
##              3  0  1 48
## Cases in matched pairs: 91.01 %
## 1 2 3
## 1 2 3
##           [,1]
## [1,] 0.9101124
```

Porównajmy boxploty wybranych cech przyrównując przydział z algorytmu, a rzeczywista klasa:



Możemy zauważyć, że algorytm wprowadzał większą róznicę między klasami 1,2,3, jednak z dużym przybliżeniem możemy uznać, że dobrze odwzorowuje odpowiednie cechy danych.

Zobaczmy jak wyglądają odpowiednie podobieństwa, średnie odległośći i obliczymy informacje o sylwetce zgodnie z danym klastrem w 3 klastrach:

```
## [1] "pierwsze 6 rekordów i ich przynależności klastrowe"
##      cluster neighbor sil_width
## [1,]       1        2 0.4078476
## [2,]       1        2 0.2013832
## [3,]       1        2 0.3645232
## [4,]       1        2 0.4222656
## [5,]       1        2 0.2137560
## [6,]       1        2 0.4393457
##   cluster size ave.sil.width
## 1       1   74          0.25
## 2       2   55          0.22
## 3       3   49          0.34
```

Ważnym aspektem jest również ilość elementów w danym podziale, jednorodność, zwartość, separacja oraz ich średnie wartości dla K=3:

```
wine.pam3$clusinfo
```

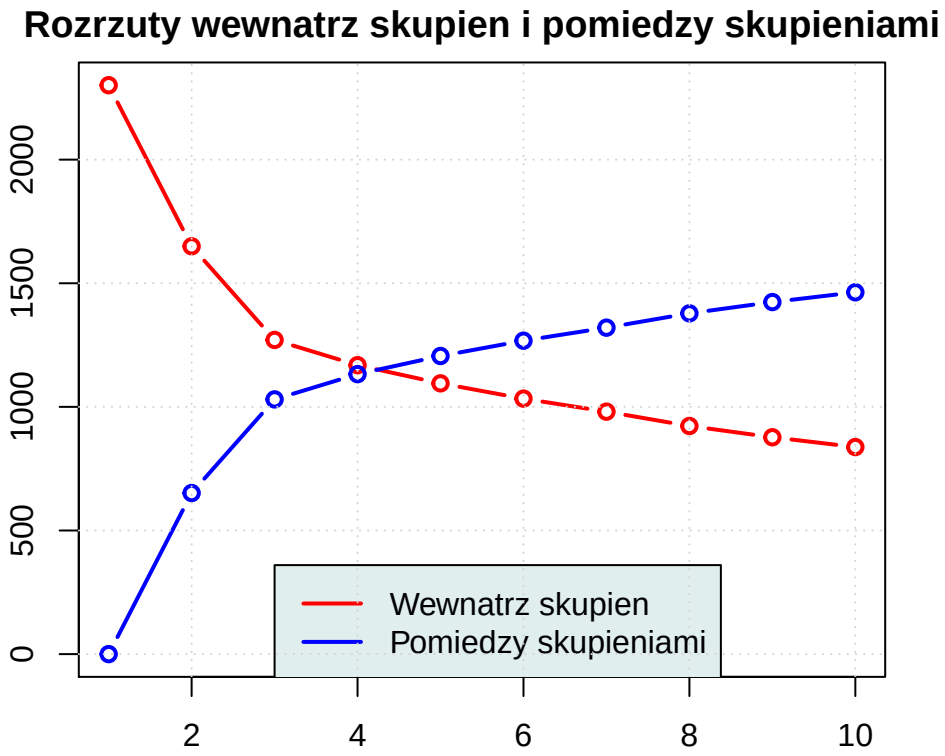
```
##      size max_diss  av_diss diameter separation
## [1,]   74 5.701178 2.876828 8.970005   1.825121
## [2,]   55 5.128242 2.899001 8.515160   1.825121
## [3,]   49 4.401335 2.595709 7.258158   2.356210
```

```
summary(wine.pam3$clusinfo)
```

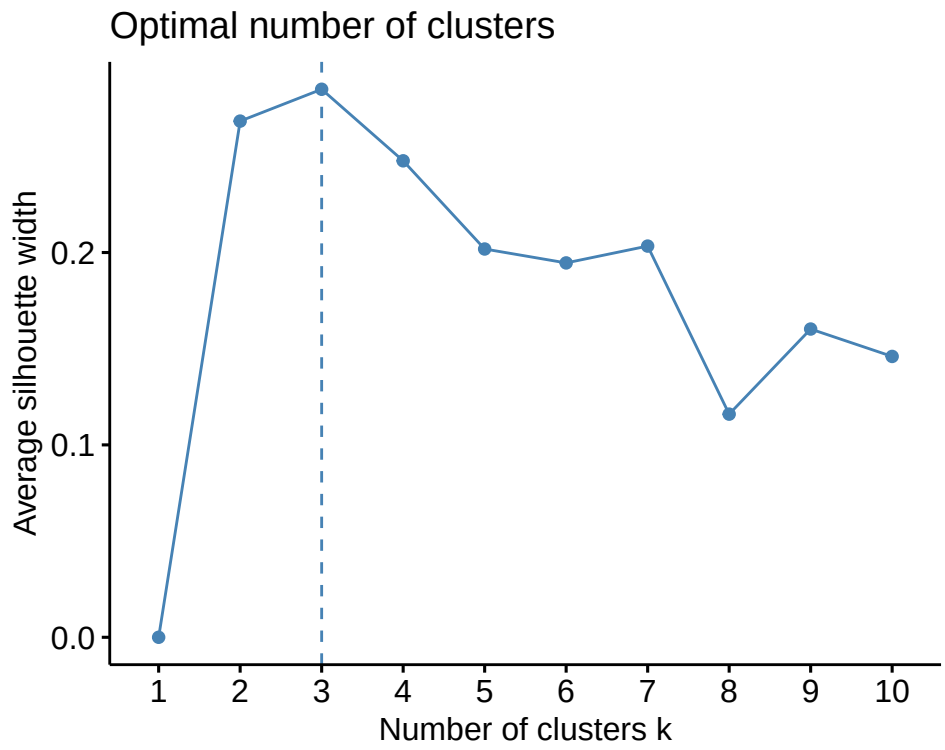
```
##       size          max_diss        av_diss         diameter
##  Min.   :49.00   Min.   :4.401   Min.   :2.596   Min.   :7.258
##  1st Qu.:52.00   1st Qu.:4.765   1st Qu.:2.736   1st Qu.:7.887
##  Median :55.00   Median :5.128   Median :2.877   Median :8.515
##  Mean   :59.33   Mean   :5.077   Mean   :2.791   Mean   :8.248
##  3rd Qu.:64.50   3rd Qu.:5.415   3rd Qu.:2.888   3rd Qu.:8.743
##  Max.   :74.00   Max.   :5.701   Max.   :2.899   Max.   :8.970
##    separation
##  Min.   :1.825
##  1st Qu.:1.825
##  Median :1.825
##  Mean   :2.002
##  3rd Qu.:2.091
##  Max.   :2.356
```

Zastanówmy się teraz, co gdybyśmy nie znali 3 typów win, a mielibyśmy za zadanie przeanalizować ten zbiór danych i stworzyć optymalną liczbę klas win, żeby odzwierciedlały jak najwięcej informacji o winach w tej klasie, pamiętając że klient nie lubi analizować zbyt wielu rodzajów win, czy istnieje lepszy z jeszcze mnijeszymi odległościami?

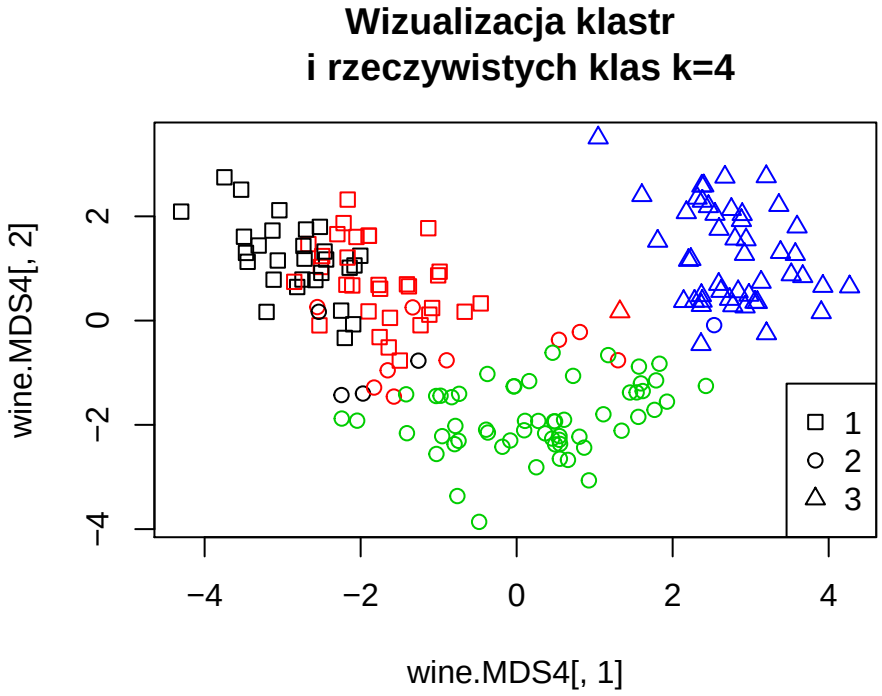
```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```



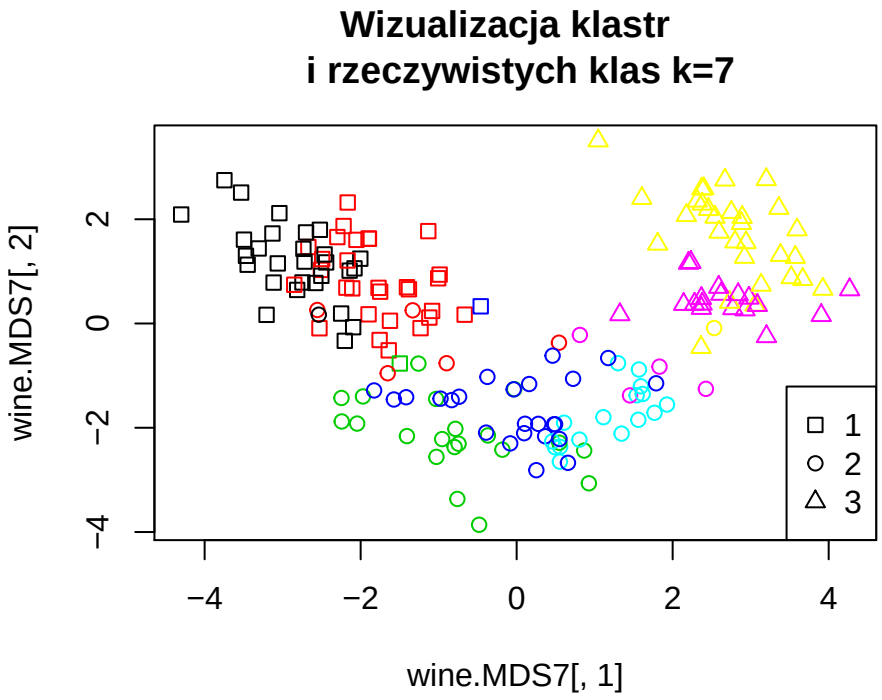
Po analizie rozrzutu możemy przyjąć że klas K powinno być coś między 3 lub 4, gdyż przy tej wartośći ich funckje zaczynają się wypłaszczać. Sprawdźmy za pomocą silhouette szukając optymalnego K:



Widzimy, że nasz model okazał się najlepszy i kolejne clustry nie przyniosą takiej różnorodnośći danych, a mogą zaburzyć zaproponowany porządek. Zweryfikujmy czy tak się dzieje np dla K = 4 oraz 7:



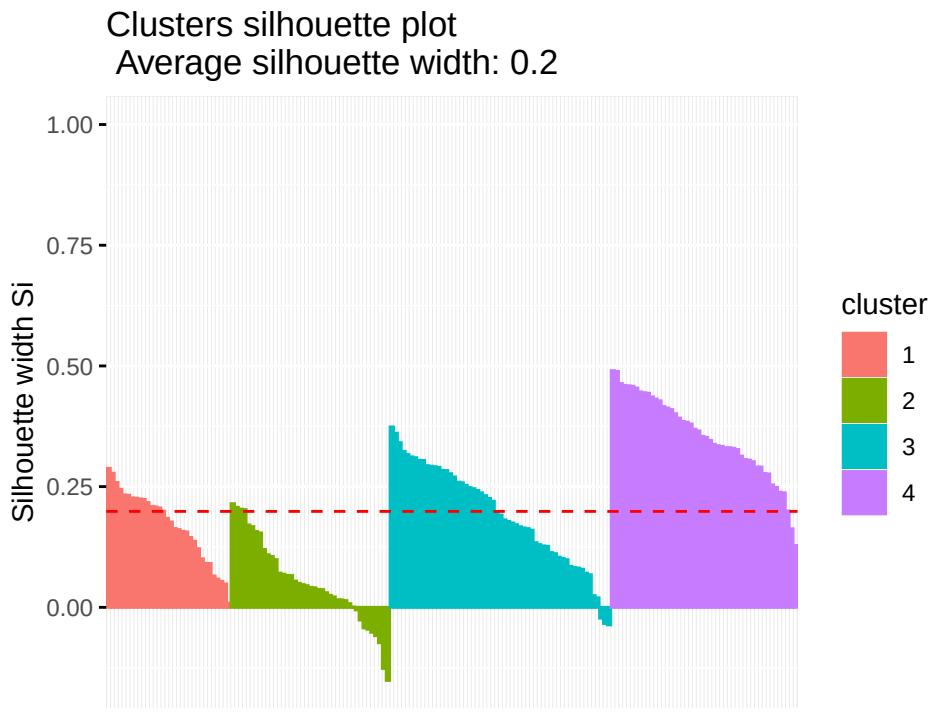
Rysunek 18: Wizualizacja



Rysunek 19: Wizualizacja

Widzimy, że powstają odrębne klastry wcześniejszych klastr. W przypadku K=4 wyodrębnia się dodatkowy podział w TYPE 1. A jak wyglądają inne parametry?
Dla K = 4:

```
sylwetka <- silhouette(wine.pam4$clustering, wina.MacNiepodob)
print("pierwsze 6 rekordów i ich przynależności klastrowe K=4")

## [1] "pierwsze 6 rekordów i ich przynależności klastrowe K=4"

sylwetka[1:6,]

##      cluster neighbor   sil_width
## [1,]       1        2  0.22470446
## [2,]       1        2  0.10071277
## [3,]       2        1 -0.04542398
## [4,]       1        2  0.23294918
## [5,]       2        1  0.20726498
## [6,]       1        2  0.20841556

fviz_silhouette(sylwetka)

##   cluster size ave.sil.width
## 1       1   32          0.17
## 2       2   41          0.05
## 3       3   57          0.19
## 4       4   48          0.35
```




```
wine.pam4$clusinfo
```

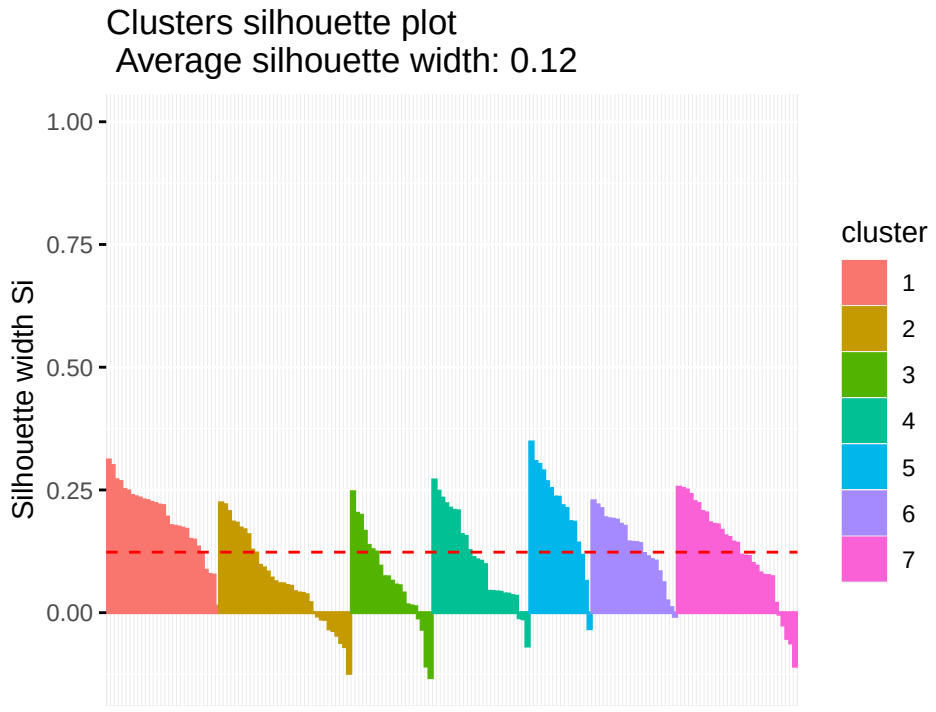
```
##      size max_diss  av_diss diameter separation
## [1,]   32 5.497864 2.501652 7.389144   1.331910
## [2,]   41 5.997600 2.651945 8.101993   1.331910
## [3,]   57 5.429670 2.915646 8.515160   1.380132
## [4,]   48 4.401335 2.561444 7.258158   2.202890
```

```
summary(wine.pam4$clusinfo)
```

```
##       size          max_diss        av_diss         diameter
##  Min.   :32.00   Min.   :4.401   Min.   :2.502   Min.   :7.258
##  1st Qu.:38.75   1st Qu.:5.173   1st Qu.:2.546   1st Qu.:7.356
##  Median :44.50   Median :5.464   Median :2.607   Median :7.746
##  Mean   :44.50   Mean   :5.332   Mean   :2.658   Mean   :7.816
##  3rd Qu.:50.25   3rd Qu.:5.623   3rd Qu.:2.718   3rd Qu.:8.205
##  Max.   :57.00   Max.   :5.998   Max.   :2.916   Max.   :8.515
##    separation
##  Min.   :1.332
##  1st Qu.:1.332
##  Median :1.356
##  Mean   :1.562
##  3rd Qu.:1.586
##  Max.   :2.203
```

Mniejsza separacja klastrowa, przynależności do klastra 2 jest bardzo losowa, nadal dostajemy dobry podział na TYPE 2,3 porównując do rzeczywistych wartości, spadek średniej wartości silhouette na poziomie 0.2, gdzie dla K=3 było to w okolicach 0.27, co dzieje się dla K=7?
K = 7

```
sylwetka <- silhouette(wine.pam7$clustering, wina.MacNiepodob)
print("pierwsze 6 rekordów i ich przynależności klastrowe K=7")

## [1] "pierwsze 6 rekordów i ich przynależności klastrowe K=7"

sylwetka[1:6,]

##      cluster neighbor   sil_width
## [1,]       1        2  0.22833336
## [2,]       1        2  0.12112272
## [3,]       2        1 -0.03315941
## [4,]       1        2  0.25117441
## [5,]       2        1  0.21995167
## [6,]       1        2  0.22469764

fviz_silhouette(sylwetka)

##   cluster size ave.sil.width
## 1       1   29          0.19
## 2       2   34          0.06
## 3       3   21          0.07
## 4       4   25          0.11
## 5       5   16          0.21
## 6       6   22          0.14
## 7       7   31          0.12
```



```
wine.pam7$clusinfo
```

```
##      size max_diss  av_diss diameter separation
## [1,]   29 5.123509 2.281491 6.691817   1.331910
## [2,]   34 5.997600 2.461539 7.642326   1.331910
## [3,]   21 5.394199 2.653567 7.099737   1.361471
## [4,]   25 4.991661 2.524530 6.468210   1.545221
## [5,]   16 3.833422 2.199124 5.407404   1.967275
## [6,]   22 3.829828 2.366390 5.846376   1.550962
## [7,]   31 4.401335 2.408669 6.467068   1.550962
```

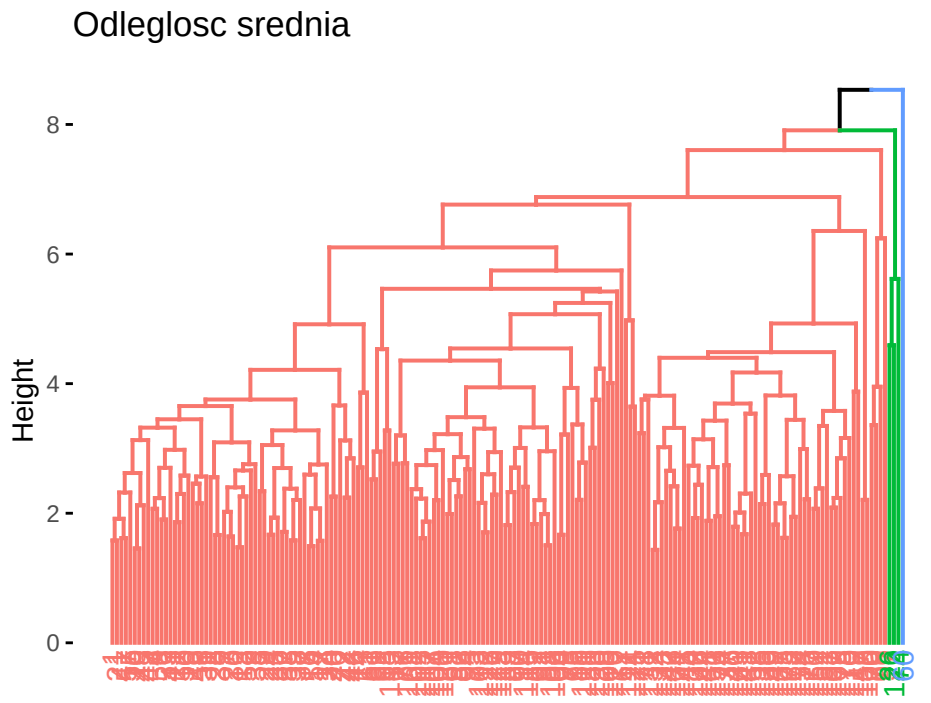
```
summary(wine.pam7$clusinfo)
```

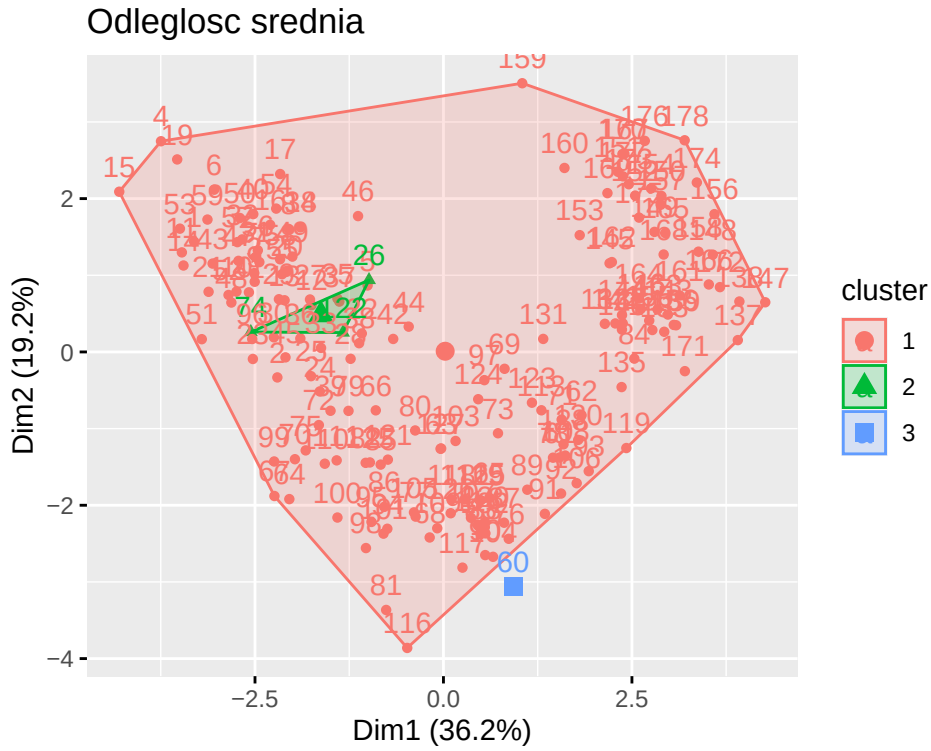
```
##       size          max_diss        av_diss         diameter
##  Min.   :16.00   Min.   :3.830   Min.   :2.199   Min.   :5.407
##  1st Qu.:21.50   1st Qu.:4.117   1st Qu.:2.324   1st Qu.:6.157
##  Median :25.00   Median :4.992   Median :2.409   Median :6.468
##  Mean   :25.43   Mean   :4.796   Mean   :2.414   Mean   :6.518
##  3rd Qu.:30.00   3rd Qu.:5.259   3rd Qu.:2.493   3rd Qu.:6.896
##  Max.   :34.00   Max.   :5.998   Max.   :2.654   Max.   :7.642
##    separation
##  Min.   :1.332
##  1st Qu.:1.347
##  Median :1.545
##  Mean   :1.520
##  3rd Qu.:1.551
##  Max.   :1.967
```

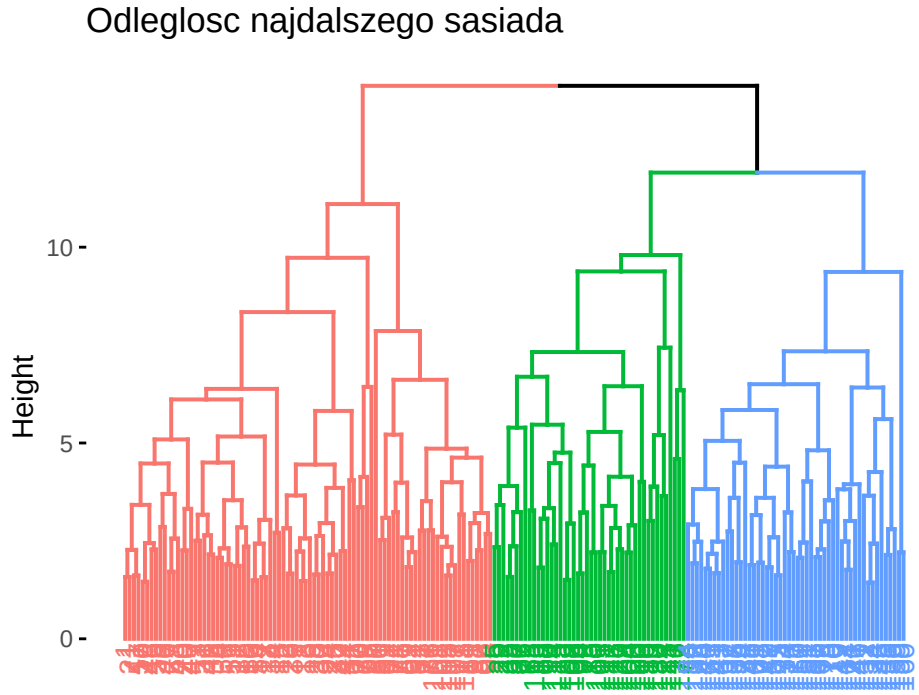
Stworzenie co najmniej po dwa podzbiory zbiorów wyjściowych, jednak gdybyśmy nie znali ich wcześniej ciężko byłoby uznać je za jedną klasę win, dlatego taki podział jest mało optymalny, dodatkowo klastry oznaczone numerami 2 i 3 mają dużą skalę błedów(ujemne wartości na wykresie), przypuszczalnie jest to pozostałość po mieszaniu się wyjściowych typów win o numerach 1 i 2. Oczywiście znaczne zmiejszenie odległości separacji i ich średnic. Dużo niższa średnia wartość silhouette na poziomie 0.12

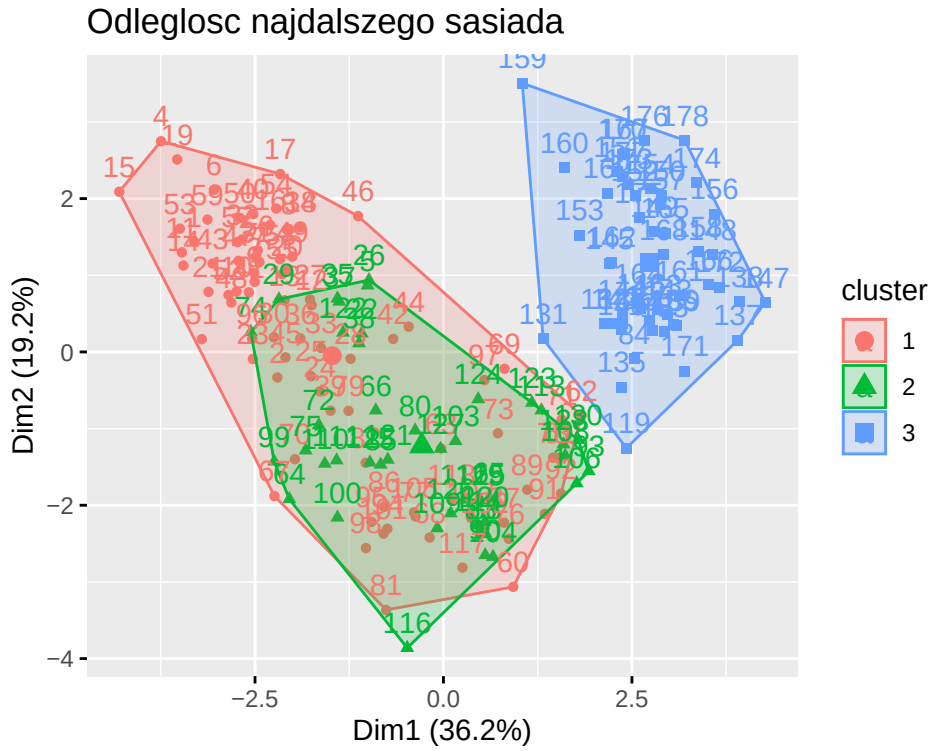
2.2 Algorytm hierarchiczny - AGNES

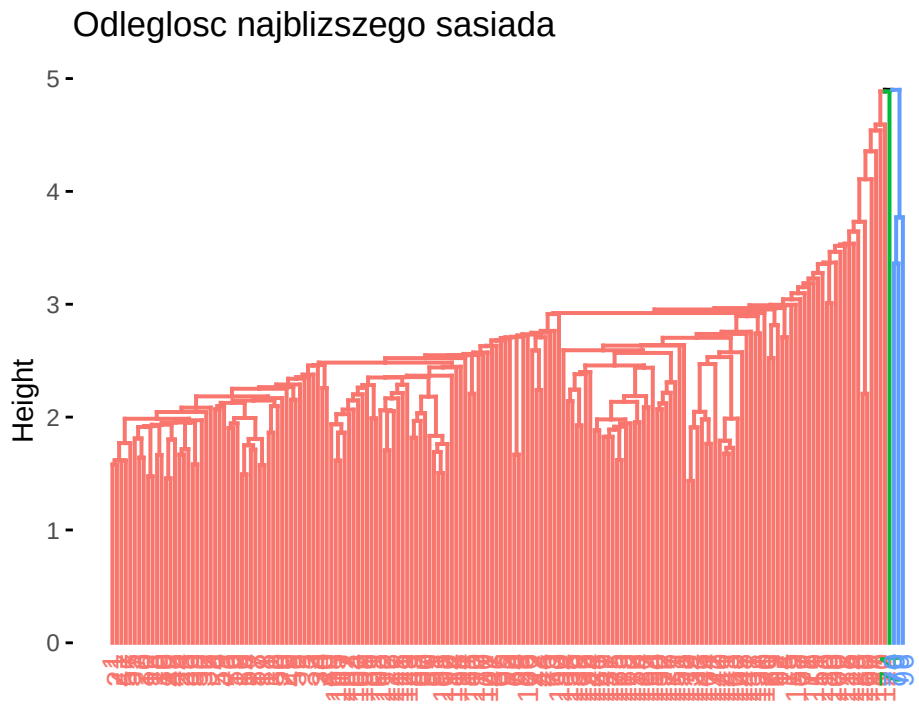
2.2.1 Wizualizacja danych

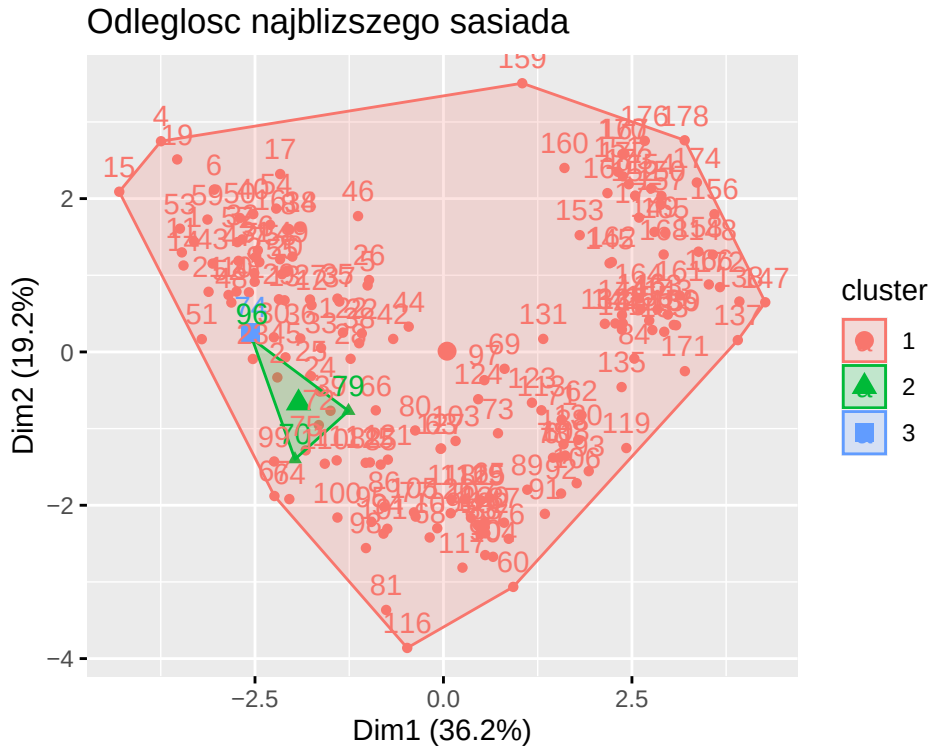

Odleglosc srednia
Height
8
6
4
2
0


Odleglosc srednia
Dim2 (19.2%)
Dim1 (36.2%)
2
0
−2
−4
−2.5
0.0
2.5
cluster
1
2
3


Odleglosc najdalszego sasiada
Height
10
5
0


Odleglosc najdalszego sasiada
Dim2 (19.2%)
Dim1 (36.2%)
2
0
−2
−4
−2.5
0.0
2.5
cluster
1
2
3





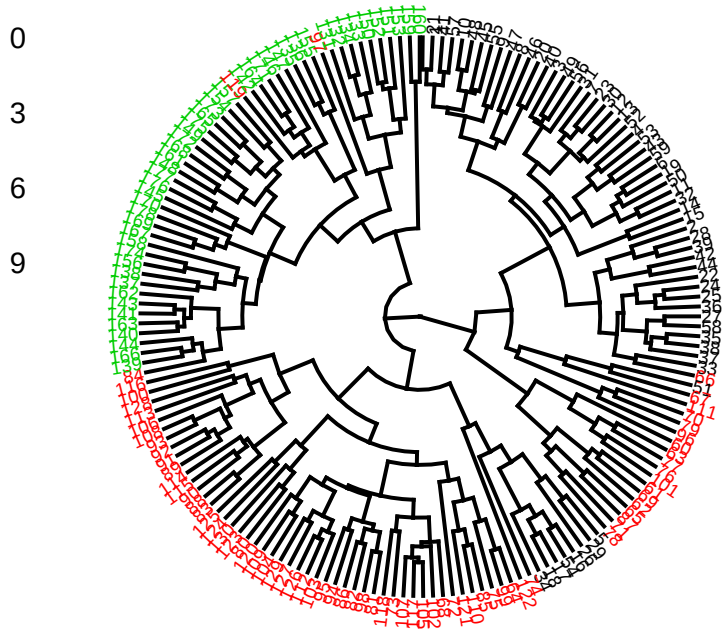
Obserwacje i wnioski:

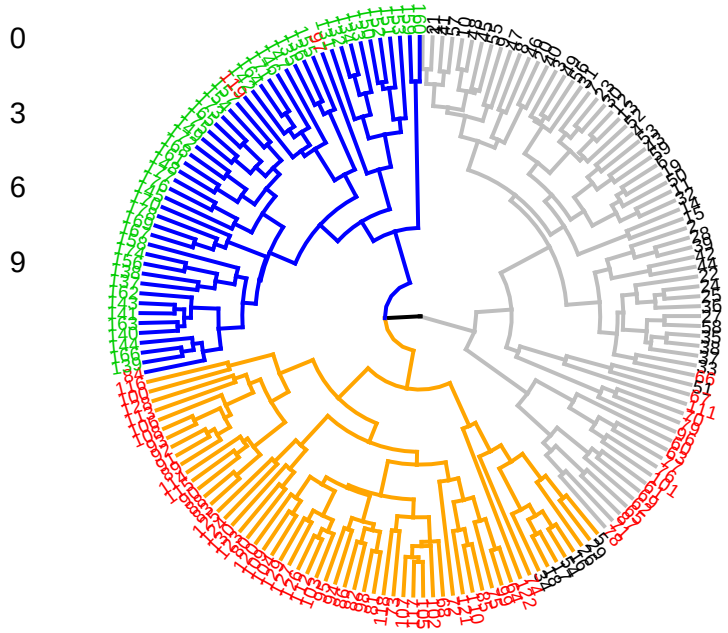
– Odległość średnia: Jedna, całkowicie dominująca klasa. Wewnątrz tego skupienia mamy 3 mniejsze, co świadczy o zwartości. Jednak proponowany podział kompletnie nie odzwierciedla faktycznego podziału na klasy, nie ma między nimi żadnego powiązania.

– Odległość najdalszego sąsiada: Wydaje się być najbardziej optymalna dla naszych danych. 3 klasy, bez wyraźnej dominacji którejkolwiek z nich. Na wykresie rozrzutu widać wyraźną separację klasy 3 od 1 i 2, które nakładają się na siebie.

– Odległość najbliższego sąsiada: Niemal identyczna sytuacja jak w przypadku odległości średniej.

Wobec powyższego w dalszej analizie będziemy się posługiwać odległością najdalszego sąsiada. Sprawdźmy rzeczywistą przynależność obiektów do klas.



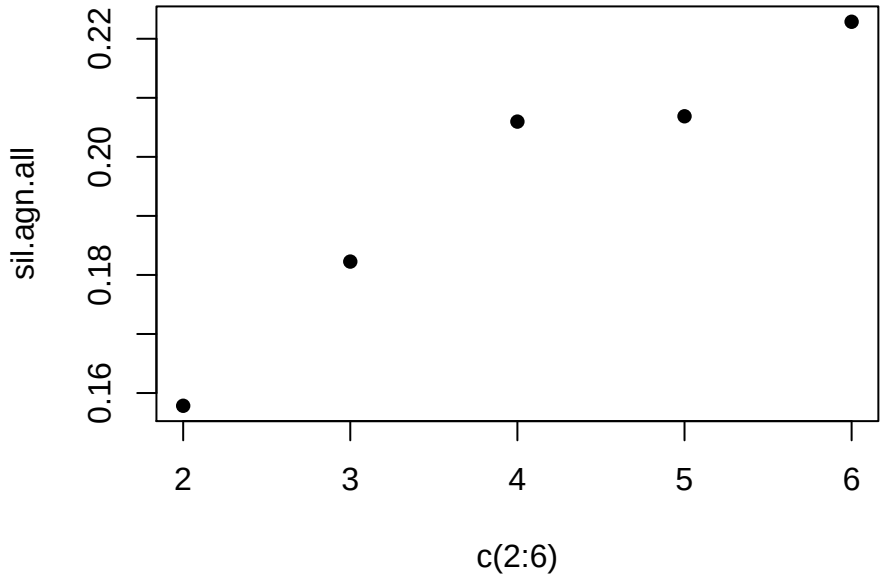


Wnioski:

- Gołym okiem widać, że większość danych została dobrze rozdzielona. Klasa 2 została w 100% wrzucona do jednego skupienia, w pozostałych dwóch popawność spada, jednak wciąż jest na doć wysokim poziomie.

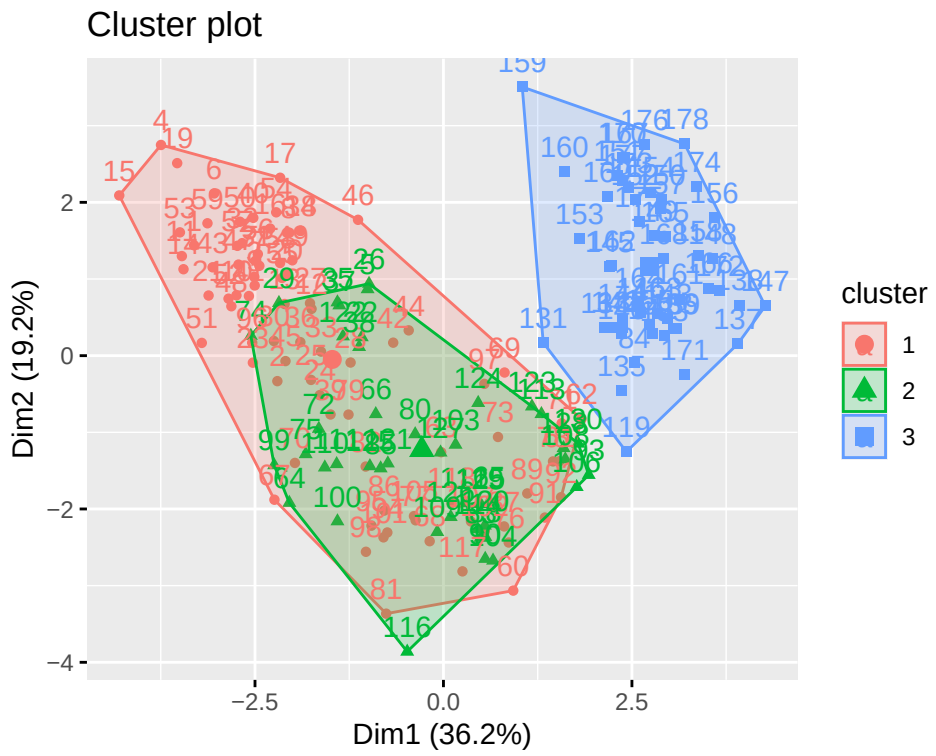
2.2.2 Wybór optymalnego K - AGNES

Wyznaczmy najpierw średnie wartości *silhouette* dla **K** należącego do zbioru od 2 do 6.



Porównajmy teraz wyniki grupowania z rzeczywistą przynależnością do klas

Podział na 3 klasy ma zgodność na poziomie ponad 80%, dzięki czemu wprowadzanie dodatkowych klas byłoby zbędnym procesem zmniejszającym przejrzystość.



Rysunek 20: Podział na skupienia dla K=2 AGNES

Ostateczne wnioski:

– Metoda PAM dla K=3 ma zgodność z rzeczywistą przynależnością do klas na poziomie 91%, co jest lepszym wynikiem niż w przypadku metody AGNES, której zgodność wynosi lekko ponad 80%.