Raport nr 3

Emilia Kowal [249716], Jakub Dworzański [249703]

17 maja 2020

Spis treści

1 Klasyfikacja na bazie modelu regresji liniowej

```
library(datasets)
library(ggplot2)
library(varhandle)
```

1.1 Krótki opis zagadnienia

W tej sekcji będziemy zajmować się klasyfikacją opartą na modelu regresji liniowej, z wykorzystaniem metody najmniejszych kwadratów. Analizy dokonujemy na zbiorze danych *iris*:

Tabela 1: Przykładowe dane

Sepal.Length	Sepal.Width	Petal.Length	Petal.Width	Species
5.1	3.5	1.4	0.2	setosa
4.9	3.0	1.4	0.2	setosa
4.7	3.2	1.3	0.2	setosa
4.6	3.1	1.5	0.2	setosa
5.0	3.6	1.4	0.2	setosa
5.4	3.9	1.7	0.4	setosa

Zbiór danych zawiera 150 wierszy z informacjami na temat 3 gatunków irysów: *setosa*, *versicolor* i *virginica*. Zmienne dostarczają nam wiedzy o szerokości i długości działki kielicha oraz płatka. Dane nie zawierają obserwacji brakujących.

Po dokonaniu analizy postaramy się odpowiedzieć na pytania o:

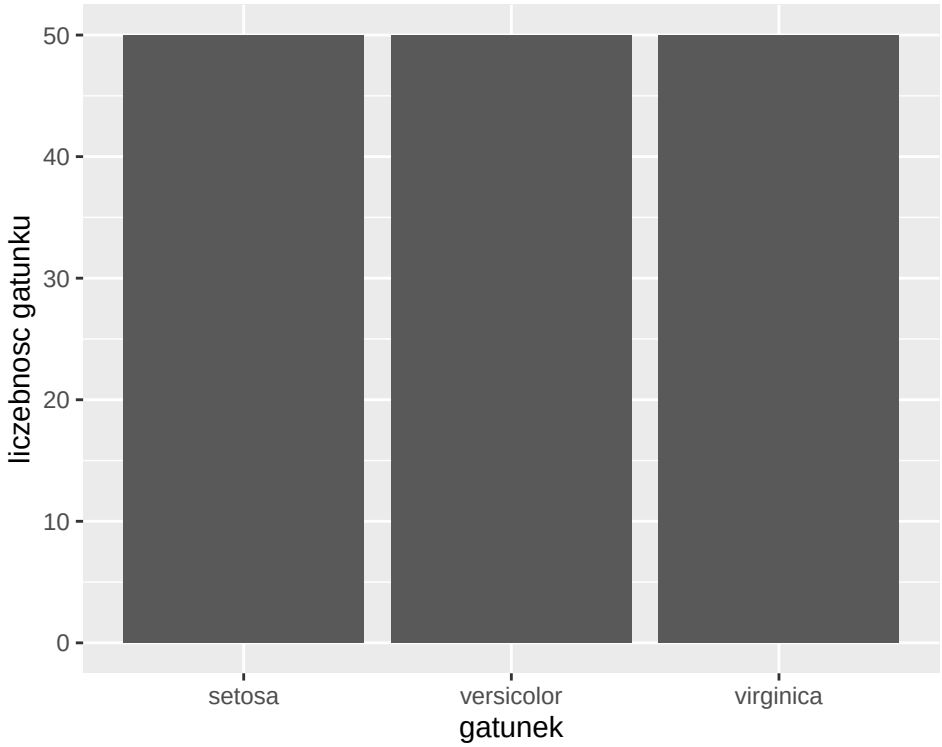
- skuteczność klasyfikacji opartej na modelu regresji liniowej;
- wpływ ilości klas na jakość predykcji;
- maskowanie klas;
- wpływ składników wielomianowych stopnia 2 na dokładność klasyfikacji.

1.2 Opis eksperymentów/analiz

- W pierwszym kroku, budując model klasyfikacyjny, dokonujemy podziału zbioru danych na treningowy oraz testowy w stosunku 1:5.
- Następnie na podstawie danych uczących kostruujemy klasyfikator i predykujemy etykiety dla zbioru treningowego i testowego.
- Potem dokonujemy oceny jakości modelu, wyznaczamy macierz pomyłek, przeprowadzamy analizę graficzną klasyfikatora.
- Ostatecznie badamy wpływ składników wielomianowych stopnia 2 na dokładność modelu regresji liniowej, dodając do zbioru danych cechę Petal.Length^2 oraz $\text{Petal.Width} * \text{Sepal.Length}$.

1.3 Wyniki

Na podstawie wykresu widzimy, iż poszczególne klasy zwierają tyle samo obiektów:



Rysunek 1: Przynależności do klas.

1.3.1 Analiza skuteczności modelu dla zbioru danych bez składników wielomianowych stopnia 2.

Dokonujemy podziału zbioru danych na trengingowe oraz testowe:

```
smp_size <- floor(0.8 * nrow(iris))
set.seed(43)
train_ind <- sample(seq_len(nrow(iris)), size=smp_size)
# pamiętamy, żeby uwzględnić wyraz wolny w modelu regresji
X_train <- cbind(rep(1,120),iris[train_ind, 1:4])
X_test <- cbind(rep(1,30),iris[-train_ind, 1:4])
Y_train <- iris[train_ind, 5]
Y_test <- iris[-train_ind, 5]
```

Konwertujemy zmienną kategoryczną na macierz wskaźnikową z wykorzystaniem funkcji *to.dummy* z pakietu *varhandle*:

```
Y_train_bin <- to.dummy(Y_train, prefix="species")
Y_test_bin <- to.dummy(Y_test, prefix="species")
```

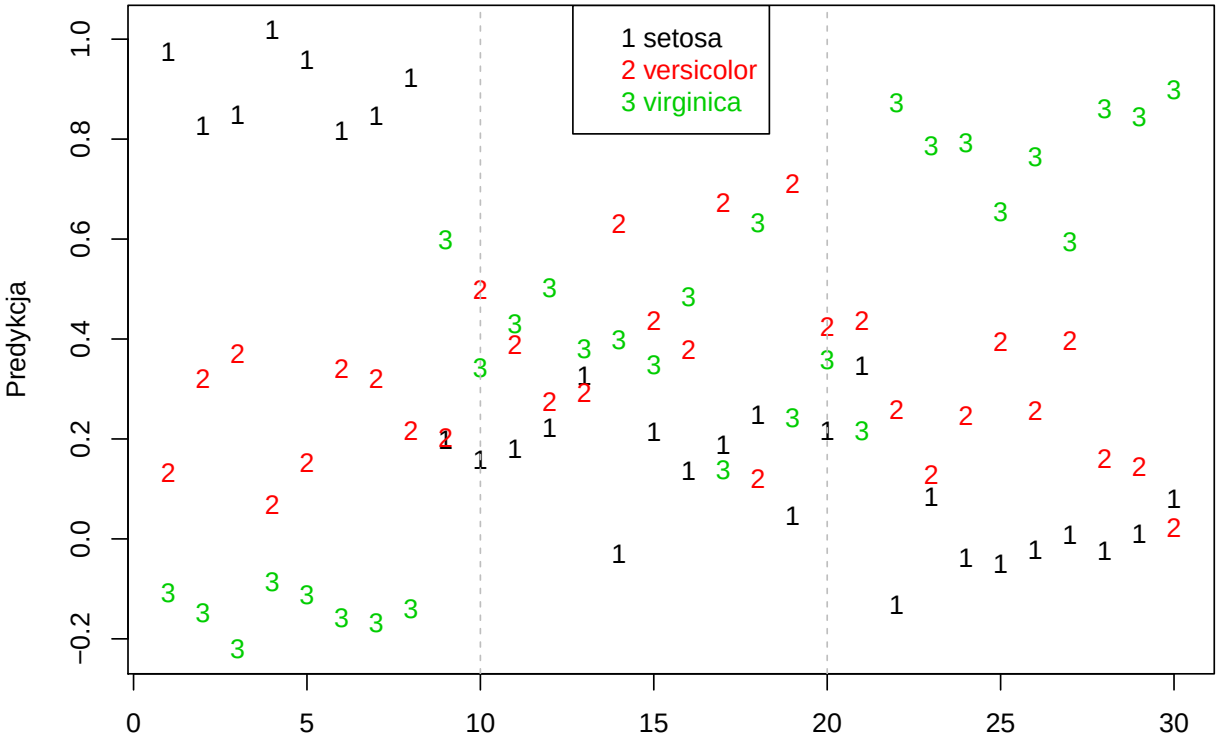
Konstrujujemy klasyfikator regresji liniowej metodą najmniejszych kwadratów:

```
B <- solve(t(X_train)%*%as.matrix(X_train)) %*% t(X_train) %*% as.matrix(Y_train_bin)
```

Tabela 2: Skuteczność estymatora dla danych testowych.

	setosa	versicolor	virginica
setosa.true	8	0	0
versicolor.true	0	7	6
virginica.true	0	0	9

Dla zbioru testowego otrzymujemy następujące wyniki: 2.
Dostajemy dokładność na poziomie: 73%.



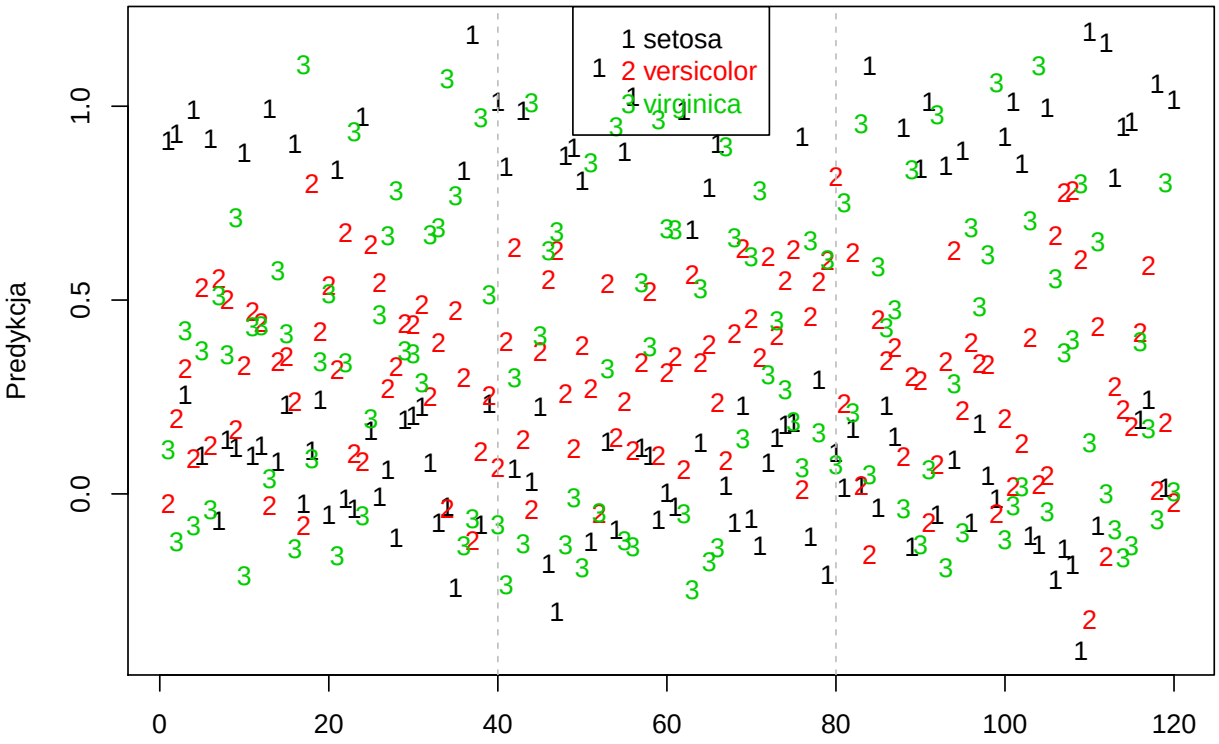
Rysunek 2: Wizualizacja wyników dla zbioru testowego.

Na wykresie 2 widzimy, że dominują klasy setosa i virginica, zatem wnioskujemy, iż versicolor jest zamaskowana (zdominowana) przez pozostałe dwie klasy.
Następnie dokonujemy analizy zgodności dla zbioru treningowego w celu uniknięcia przetrenowania modelu. Otrzymujemy następujące rezultaty: 3.

Dostajemy zgodność na poziomie: 88%, co mogłoby wskazywać na to, iż model jest przetrenowany. Aby uniknąć takiej sytuacji, w dalszej analizie należałoby przeprowadzić np. cross validation test, który dałby nam rzetelniejsze wyniki dla całego zbioru danych. Również fakt, iż w zbiorze nie mamy wielu rekordów, działa niekorzystnie na skuteczność modelu regresji.
Wyniki przedstawione graficznie:

Tabela 3: Skuteczność estymatora dla danych treningowych.

	setosa	versicolor	virginica
setosa.true	42	0	0
versicolor.true	0	25	12
virginica.true	0	5	36



Rysunek 3: Wizualizacja wyników dla zbioru treningowego.

1.3.2 Analiza skuteczności modelu dla zbioru danych ze składnikami wielomianowymi stopnia 2.

Do konstrukcji drugiego modelu regresji wykorzystujemy zbiór z wyjściowymi cechami oraz zmiennymi wielomianowymi stopnia 2.

Dla estymatora, skonstruowanego w oparciu o nowy zbiór danych, dla zbioru testowego otzymujemy następujące wyniki: 4.

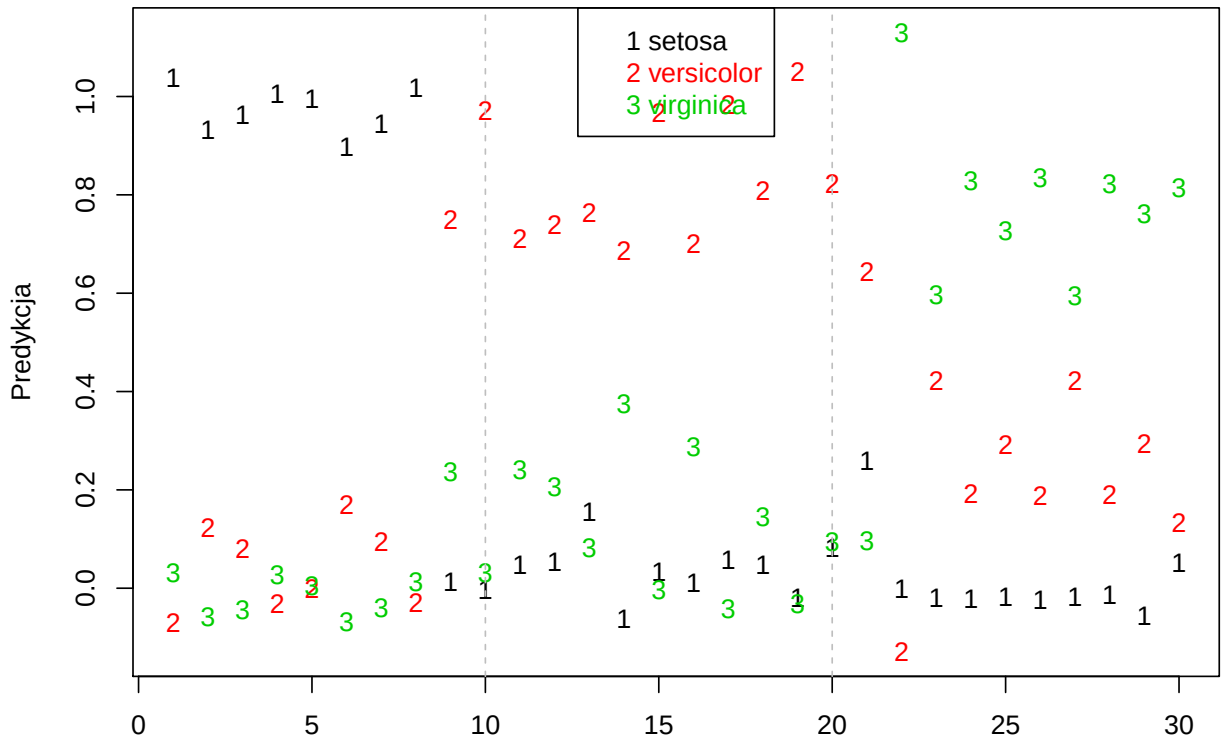
Dostajemy zgodność na poziomie: 97%, zatem otrzymujemy znacznie skuteczniejszy estymator dla zbioru danych ze składnikami wielomianowymi stopnia 2.

Tabela 4: Skuteczność nowego estymatora dla zbioru testowego.

	setosa	versicolor	virginica
setosa.true	8	0	0
versicolor.true	0	13	0
virginica.true	0	0	9

Tabela 5: Skuteczność nowego estymatora na zbiorze treningowym.

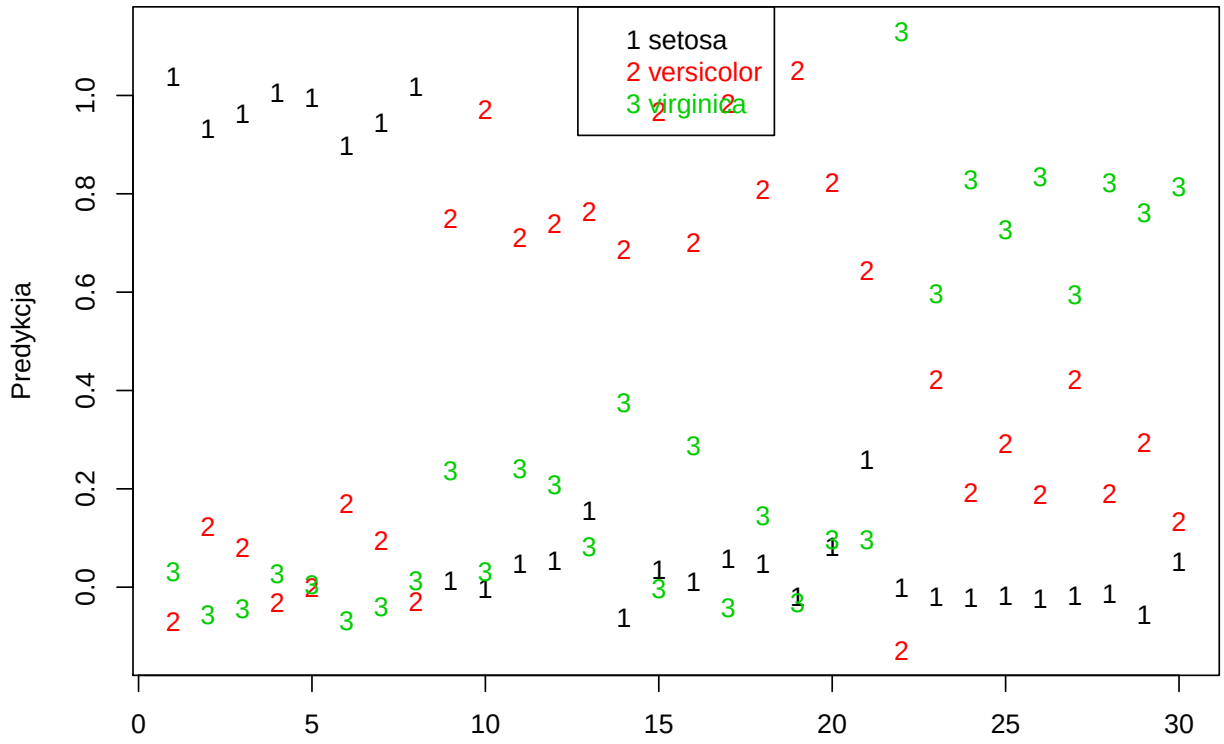
	setosa	versicolor	virginica
setosa.true	42	0	0
versicolor.true	0	36	1
virginica.true	0	2	39



Rysunek 4: Wizualizacja wyników nowego estymatora dla zbioru testowego.

Na podstawie wykresu 4 zauważamy, że dla nowego zbioru danych, klasy setosa oraz virginica nie dominują w sposób znaczny klasy versicolor. Wnioskujemy, że z tego też względu otrzymujemy znacznie lepszy rezultat dla modelu regresji.
Zbadamy jeszcze skuteczność estymatora dla zbioru treningowego: 5.

Otrzymujemy zgodność na poziomie 98%.



1.4 Podsumowanie

- Po przeprowadzeniu analiz wnioskujemy, iż model oparty na regresji liniowej jest skuteczny, kiedy zależności między zmiennymi są liniowe. Jak wiadomo, zazwyczaj jednak mamy styczność ze zmiennymi o zależnościach nieliniowych, gdzie estymator regresji może dawać niesatysfakcjonujące rezultaty. Widzimy, iż po dodaniu do zbioru danych cech wielomianowych stopnia 2, skuteczność klasyfikatora znacznie wzrasta

- Ponadto, kiedy tworzymy model klasyfikacyjny dla większej liczby klas, regresja liniowa jest narażona na problem maskowania klas, który również wpływa niekorzystnie na wyniki estymatora.

- Tworząc model regresji należy pamiętać o zapobiaganiu jego przetrenowania wykorzystując metody takie, jak *cross-validation* lub *bagging*.

Podsumowując regresja liniowa zakłada, iż w zbiorze zmienne mają liniowe zależności. Pracując nad bardziej złożonym problemem klasyfikacyjnym, dla dużej liczby klas, wybór odpowiednich składników wielomianowych może być czasochłonne, a przez to nieefektywne.

Tabela 6: Przykładowe dane

RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	Type
1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0	0.00	1
1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0	0.00	1
1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0	0.00	1
1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0	0.00	1
1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0	0.00	1
1.51596	12.79	3.61	1.62	72.97	0.64	8.07	0	0.26	1

2 Porównanie metod klasyfikacji

2.1 Krótki opis zagadnienia

W tym ćwiczeniu, będziemy korzystali ze zbioru danych o odłamkach szkła.

```
data(Glass)
head(Glass)
```

Dane składają się z informacji, dotyczących 214 odłamków. Każdy z rekordów składa się ze współczynnika załamania światła (`RI`) oraz stężenia różnych pierwiastków (w odpowiadających tlenkach) w badanych próbkach. Oprócz tego, posiadamy informacje o typie szkła, z którego odłamkiem mamy do czynienia. Razem, mamy 7 takich kas, z których tylko 6 występuje w danych. Łącznie, dla każdej próbki występuje 10 cech. Ponadto, w zbiorze nie ma wartości brakujących.

Naszym celem będzie konstrukcja klasyfikatora, który będzie przyporządkowywał odłamki szkła do odpowiednich klas.

2.2 Opis eksperymentów/analiz

Skorzystamy z najczęściej stosowanych, podstawowych metod. Będą to metoda k-najbliższych sąsiadów (k-NN, *k-Nearest Neighbours*), naiwny klasyfikator bayesowski (NBC, *Naive Bayesian Classifier*) oraz drzewo klasyfikacyjne (*classification tree*).

Ponadto, postaramy się dobrać odpowiednie parametry dla tych modeli. Spróbujemy również dokonać pewnych przekształceń danych, czy też wybrać pewne podzbiory cech, aby sprawdzić, czy uda nam się w ten sposób poprawić jakość klasyfikatora.

Po zbadaniu podstawowych metod, spróbujemy również porównać je z metodami bardziej zaawansowanymi i wyrafinowanymi.

Aby ocenić jakość predykcji, będziemy korzystać z 5-, 10-krotnego sprawdzianu krzyżowego oraz sprawdzianu krzyżowego typu leave-one-out (*5-, 10-, leave-one-out Cross Validation*). Ponadto, wykorzystamy metodę *bootstrap* oraz *632+*.

Zastosujemy do tego następujące funkcje pomocnicze, odpowiadające za uczenie modelu oraz estymację błędów.

```
mypredict <- function(model, newdata) predict(model, newdata, type="class")
nasz.errorest <- function(formula, data, model, ...){
  cv.10 <- errorest(
    formula=formula, data=data, model=model, predict=mypredict,
```

```
    estimator="cv", est.para=control.errorest(k = 10), ...
  )$error
  cv.5 <- errorest(
    formula=formula, data=data, model=model, predict=mypredict,
    estimator="cv", est.para=control.errorest(k = 5), ...
  )$error
  leave.one.out <- errorest(
    formula=formula, data=data, model=model, predict=mypredict,
    estimator="cv", est.para=control.errorest(k = nrow(data)), ...
  )$error
  bootstrap <- errorest(
    formula=formula, data=data, model=model,
    predict=mypredict, estimator="boot", ...
  )$error
  err632plus <- errorest(
    formula=formula, data=data, model=model,
    predict=mypredict, estimator="632plus", ...
  )$error
  return(data.frame(
    paste0(
      round(100*c(cv.5, cv.10, leave.one.out, bootstrap, err632plus), 2),
      "%"
    ),
    row.names=c(
      "CV5", "CV10", "CV leave-one-out",
      "bootstrap (25 powtórzeń)",
      "632+ (25 powtórzeń)"
    )
  ))
}
```

2.3 Wyniki

2.3.1 Modele z domyślnymi wartościami dla wszystkich cech

Nasze badania rozpoczniemy od estymacji błędu klasyfikacji dla modeli z domyślnymi parametrami i bez wcześniejszych przekształceń danych. Pozwoli to nam później ocenić ewentualną poprawę wyników.

```
wyniki <- nasz.errorest(Type ~ ., Glass, ipredknn)
wyniki["drzewo"] <- nasz.errorest(Type ~ ., Glass, rpart)
wyniki["nb"] <- nasz.errorest(Type ~ ., Glass, naiveBayes)

wyniki
```

W tabeli 7 widzimy, że dla domyślnych parametrów, z całymi danymi najlepiej radzą sobie metoda k-najbliższych sąsiadów oraz drzewa klasyfikacyjne. Można również zauważyć, że naiwny klasyifkator bayesowski zadziałał w tym przypadku zdecydowanie gorzej.

Tabela 7: Porównanie błędu klasyfikacji dla modeli z domyślnymi parametrami.

	k-NN	drzewo klasyfikacyjne	naiwny klasyfikator bayesowski
CV5	30.37%	33.18%	63.08%
CV10	33.64%	33.18%	62.15%
CV leave-one-out	32.24%	28.04%	57.48%
bootstrap (25 powtórzeń)	36.58%	35.72%	57.08%
632+ (25 powtórzeń)	31.38%	31.19%	57.61%

Tabela 8: Sprawdzenie poprawy błędu klasyfikacji po zmianie parametru k.

	k=1	k=2	k=3	k=4	k=5	k=10	k=15
CV5	28.97%	31.31%	30.37%	35.98%	30.37%	41.12%	40.65%
CV10	27.57%	31.78%	31.78%	34.11%	33.64%	36.45%	38.79%
CV leave-one-out	26.64%	29.44%	32.24%	33.18%	32.24%	36.92%	36.92%
bootstrap (25 powtórzeń)	31.23%	33.01%	34.9%	36.42%	36.58%	36.75%	38.04%
632+ (25 powtórzeń)	23.01%	28.08%	29.7%	30.89%	31.38%	35.38%	37.09%

Postaramy się znaleźć odpowiednie podzbiory cech oraz parametry dla poszczególnych modeli, aby poprawić otrzymane rezultaty.

2.3.2 Analiza opisowa

2.3.3 Metoda k-najbliższych sąsiadów

Sprawdzimy, jak możemy zmienić jakość modelu k-NN, zmieniając wartość k, która reprezentuje liczbę sąsiadów, na podstawie których wybieramy klasę.

```
knn <- function(k) partial(ipredknn, k=k)
wyniki["k.nn.1"] <- nasz.errorest(Type ~ ., Glass, knn(1))
wyniki["k.nn.2"] <- nasz.errorest(Type ~ ., Glass, knn(2))
wyniki["k.nn.3"] <- nasz.errorest(Type ~ ., Glass, knn(3))
wyniki["k.nn.4"] <- nasz.errorest(Type ~ ., Glass, knn(4))
wyniki["k.nn.10"] <- nasz.errorest(Type ~ ., Glass, knn(10))
wyniki["k.nn.15"] <- nasz.errorest(Type ~ ., Glass, knn(15))

wyniki
```

Na podstawie tabeli 8 widzimy, że istotnie możemy znacznie poprawić wyniki, zmieniając wyłącznie parametr k. Ponadto, widzimy, że w porównaniu do pozostałych wartości, model bardzo dobrze działa dla k=1.

Aby poprawić wyniki, możemy jeszcze spróbować ograniczyć liczbę zmiennych, stosując PCA.

```
Glass.pca <- prcomp(subset(Glass, select=-c(Type)), retx=T, center=T, scale.=T)
Glass.pca <- data.frame(Glass.pca$x[,1:4])
Glass.pca$Type <- Glass$Type
```

Tabela 9: Sprawdzenie poprawy błędu klasyfikacji po użyciu pierwszych 4 głównych składowych

	k=1	k=2	k=3	k=4	k=5	k=10	k=15
CV5	28.04%	29.91%	34.11%	31.78%	27.1%	34.11%	36.45%
CV10	30.37%	33.18%	30.84%	28.5%	28.5%	35.05%	35.98%
CV leave-one-out	29.91%	30.84%	32.24%	29.44%	28.97%	31.78%	33.18%
bootstrap (25 powtórzeń)	31.7%	33.03%	33.72%	35.19%	35.18%	34.64%	37.19%
632+ (25 powtórzeń)	23.61%	29.25%	28.99%	30.62%	30.77%	31.46%	34.58%

```
wyniki["k.nn.1.pca"] <- nasz.errorest(Type ~ ., Glass.pca, knn(1))
wyniki["k.nn.2.pca"] <- nasz.errorest(Type ~ ., Glass.pca, knn(2))
wyniki["k.nn.3.pca"] <- nasz.errorest(Type ~ ., Glass.pca, knn(3))
wyniki["k.nn.4.pca"] <- nasz.errorest(Type ~ ., Glass.pca, knn(4))
wyniki["k.nn.5.pca"] <- nasz.errorest(Type ~ ., Glass.pca, ipredknn)
wyniki["k.nn.10.pca"] <- nasz.errorest(Type ~ ., Glass.pca, knn(10))
wyniki["k.nn.15.pca"] <- nasz.errorest(Type ~ ., Glass.pca, knn(15))

wyniki
```

W tabeli 9 widzimy, że po redukcji danych do 4 pierwszych głównych składowych (wyjaśniają one około 80% wariancji), wyniki są porównywalne z tymi bez redukcji danych.

Widzimy zatem, że najlepsze wyniki dla metody k-NN otrzymaliśmy dla k=1.

2.3.4 Drzewo klasyfikacyjne

Skorzystamy z metody jednego odchylenia standardowego, przedstawionej na wykładzie, aby zmniejszać złożoność drzewa po uczeniu.

```
nasze.drzewo <- function(...){
  drzewo <- rpart(..., minsplit=1)
  cptable <- data.frame(drzewo$cptable)
  xerror.min <- min(cptable$xerror)
  xstd.min <- max(cptable[which(cptable$xerror == xerror.min),]$xstd)
  ponizej.1.sd <- cptable[which(cptable$xerror < xerror.min + xstd.min),]$CP
  return(prune(drzewo, cp=max(ponizej.1.sd)))
}
wyniki["przyciete.drzewo"] <- nasz.errorest(Type ~ ., Glass, nasze.drzewo)
wyniki[c("drzewo", "przyciete.drzewo")]
```

W tabeli 10 widzimy, że przycięcie drzewa za pomocą metody 1 odchylenia standardowego pozwala nie zmienia znacząco wyników. Możemy jednak przypuszczać, że przycięte drzewo ma lepsze właściwości uogólniające.

Możemy również spróbować wybrać najważniejsze zmienne, tworząc drzewo klasyfikacyjne, a następnie skonstruować model z wykorzystaniem tych cech.

```
drzewo <- rpart(Type~., Glass, minsplit=1)
drzewo$variable.importance
```

Tabela 10: Sprawdzenie poprawy błędu klasyfikacji po przycięciu drzewa

	drzewo	przycięte drzewo
CV5	33.18%	35.51%
CV10	33.18%	28.5%
CV leave-one-out	28.04%	30.37%
bootstrap (25 powtórzeń)	35.72%	34.46%
632+ (25 powtórzeń)	31.19%	31.68%

Tabela 11: Ważność zmiennych na podstawie konstrukcji drzewa klasyfikacyjnego

	Ważność zmiennej
RI	46.621949
Al	41.239493
Ca	39.191590
Mg	38.944662
Na	27.423009
Ba	26.419293
K	23.641254
Si	21.378069
Fe	8.000758

Na podstawie otrzymanych ważności zmiennych (tab.11), spróbujemy skonstruować dwa modele, ograniczając ilość zmiennych w modelu.

```
wyniki["drzewo1"] <- nasz.errorest(
  Type ~ RI + Al + Ca, Glass, rpart
)
wyniki["drzewo2"] <- nasz.errorest(
  Type ~ RI + Al + Ca + Mg + Na + Ba, Glass, rpart
)
wyniki["drzewo1.przyciete"] <- nasz.errorest(
  Type ~ RI + Al + Ca, Glass, nasze.drzewo
)
wyniki["drzewo2.przyciete"] <- nasz.errorest(
  Type ~ RI + Al + Ca + Mg + Na + Ba, Glass, nasze.drzewo()
)

wyniki
```

Na podstawie wyników z tabeli 12, widzimy, że spośród tych modeli, najlepiej sprawdza się model ograniczony do 6 najważniejszych zmiennych, który jest przycinany po konstrukcji. Ponadto, możemy zauważyć, że te wyniki są lepsze niż wstępne wyniki(tab.10).

2.3.5 Naiwny klasyfikator bayesowski

W tej części postaramy się poprawić wyniki naiwnego klasyfikatora bayesowskiego.

Tabela 12: Wyniki dla drzew z ograniczoną ilością zmiennych niezależnych

	drzewo 1	przycięte drzewo 1	drzewo 2	przycięte drzewo 2
CV5	35.51%	35.05%	29.44%	34.58%
CV10	32.24%	34.58%	33.64%	28.97%
CV leave-one-out	34.11%	36.45%	28.5%	30.84%
bootstrap (25 powtórzeń)	36.52%	35.15%	32.78%	30.94%
632+ (25 powtórzeń)	34.17%	32.55%	30.67%	29.69%

Tabela 13: Porównanie wyników naiwnego klasyfikatora bayesowskiego z użyciem estymacji gęstości rozkładu

	rozkład normalny	estymacja gęstości rozkładu
CV5	63.08%	41.59%
CV10	62.15%	46.73%
CV leave-one-out	57.48%	41.59%
bootstrap (25 powtórzeń)	57.08%	43.41%
632+ (25 powtórzeń)	57.61%	39.02%

Wszystkie zmienne w naszym zbiorze to zmienne ciągłe. Domyślnie, implementacja naiwnego klasyfikatora bayesowskiego dokonuje założenia o rozkładzie normalnym zmiennych ciągłych. Dlatego też rozpoczniemy od zmiany tego założenia na estymację gęstości rozkładów zmiennych ciągłych.

```
wyniki["nb.kernel"] <- nasz.errorest(Type~., Glass, naive_bayes, usekernel=T)
wyniki[c("nb", "nb.kernel")]
```

W tabeli 13 widzimy, że prosta zmiana pozwoliła na znaczną poprawę wyników.

2.3.6 Komitety klasyfikatorów

Po zbadaniu i próbie poprawy klasyfikatorów k-NN, drzew klasyfikacyjnych oraz naiwnego klasyfikatora bayesowskiego, spróbujemy wykorzystać komitety klasyfikatorów, aby sprawdzić, czy otrzymamy w ten sposób porównywalne wyniki.

Wykorzystamy w tym celu las losowy.

```
wyniki["random.forest"] <- nasz.errorest(Type~., Glass, randomForest)
wyniki
```

W tabeli 14 widzimy, że, nawet dla domyślnych parametrów, las losowy poradził sobie z danymi najlepiej spośród przygotowanych przez nas klasyfikatorów. Stąd, możemy przypuszczać, że poprawa parametrów, mogłaby jeszcze bardziej zwiększyć różnicę między tymi klasyfikatorami.

2.4 Podsumowanie

Udało nam się skonstruować działające klasyfikatory. Warto zaznaczyć, że klasyfikator, który wskazywałby losową klasę, miałby błąd predykcji na poziomie 83%, natomiast taki, który wskazywałby zawsze klasę 2 (najliczniejszą), mógłby osiągnąć błąd na poziomie 64%.

Tabela 14: Porównanie najlepszych wyników

	NBC	k-NN	drzewo klasyfikacyjne	las losowy
CV5	41.59%	28.97%	34.58%	21.5%
CV10	46.73%	27.57%	28.97%	20.56%
CV leave-one-out	41.59%	26.64%	30.84%	20.09%
bootstrap (25 powtórzeń)	43.41%	31.23%	30.94%	25.15%
632+ (25 powtórzeń)	39.02%	23.01%	29.69%	17.21%

Nam, natomiast, udało się osiągnąć błąd predykcji rzędu 20%, co wydaje się być dość dobrym wynikiem.

Ponadto, dla każdego z podstawowych klasyfikatorów, udało nam się poprawić jego jakość. Wykorzystaliśmy do tego zmianę parametrów, ale też redukcję wymiaru, czy też ilości cech. Ponadto korzystaliśmy z metod, które zapobiegają przeuczeniu modelu, takich jak przycinanie drzewa.

Spośród najprostszych klasyfikatorów, najlepsze rezultaty otrzymaliśmy dla metody k-najbliższych sąsiadów. Wyniki dla drzewa klasyfikacyjnego były nieznacznie gorsze. Nie udało nam się jednak zbliżyć do tych wyników dla naiwnego klasyfikatora bayesowskiego. Możemy przypuszczać, że jest to związane z tym, że mamy do czynienia przede wszystkim z cechami ciągłymi, których rozkłady nie są znane.

Na zakończenie, należy zapamiętać, że korzystając z nowszych, bardziej zaawansowanych i wyrafinowanych algorytmów, takich jak las losowy, byliśmy w stanie osiągnąć rewelacyjne rezultaty, które są zdecydowanie lepsze niż wyniki dla podstawowych klasyfikatorów.