Akademia Ekonomiczno-Humanistyczna w Warszawie

SPRAWOZDANIE

INTELIGENTNA ANALIZA DANYCH

LAB2
WALIDACJA KRZYŻOWA, STRATYFIKACJA,
DRZEWA LOSOWE I NAIWNY KLASYFIKATOR bayesowski

07.11.2021

JOANNA PRAJZENDANC
36358

MIŁOSZ SAKOWSKI
36381

Spis treści

1. Cel i przebieg ćwiczenia

Celem ćwiczenia było utrwalenie wiedzy w zakresie klasyfikatorów. W pierwszej kolejności należało zapoznać się z problemem niezbalansowanego zbioru danych i sposobów rozwiązania go za pomocą walidacji krzyżowej i stratyfikacji w przypadku klasyfikatora binarnego. Następnym krokiem było zapoznanie się z klasyfikatorem drzewa losowego oraz naiwnym klasyfikatorem bayesowski.

2. Definicje i założenia

2.1. Wyjaśnienie pojęć

W sprawozdaniu pojawiają się następujące pojęcia:

- ✧ niezbalansowana baza danych: zbiór danych, w którym występuje wyraźna dysproporcja jeśli chodzi o ilość elementów dla każdej z klas,
- ✧ walidacja krzyżowa: zbiór początkowy dzielony jest na kilka podzbiorów (w zależności od parametrów podanych przez użytkownika); następnie część zbiorów jest jako zbiór trenujący, a część jako zbiór testujący; potem następuje zmiana i inna część zbiorów jest zbiorem testującym itd.
- ✧ stratyfikacja: zbiory trenujące i testujące są dobierane tak, aby mieć pewność że w każdym z podzbiorów znajdą się elementy każdej możliwej klasy,
- ✧ klasyfikator binarny: prosty podział na zbiór trenujący i testujący w ustalonych proporcjach np. 10% danych zbiór testujący i 90% zbiór trenujący,
- ✧ klasyfikator lasu losowego: las losowy składa się z kilku drzew decyzyjnych, które powstają na 3 sposoby:
  - ➢ tylko część pierwotnego zbioru jest dzielona pomiędzy zbiór testujący i zbiór trenujący, tak aby każde drzewo decyzyjne powstało i testowało na innym zbiorze danych,
  - ➢ dla każdego drzewa decyzyjnego wymuszany jest warunek, np. w pierwszym kroku użyj innego parametru,
  - ➢ połączenie dwóch powyższych zasad.
- ✧ naiwny klasyfikator bayesowski: dopasowanie elementu jest przeprowadzane na podstawie prawdopodobieństwa *a prori*.

2.2. Przygotowanie bazy danych

Podczas wykonywania zadań omówionych w sprawozdaniu, do rozwiązania użyto tej samej bazy danych z informacjami o pożarach w parku XX.

i. Importowanie bazy danych i oryginalna zawartość

Dane wykorzystane w zadaniach pochodzą z pliku forestfires.csv, który został pobrany ze strony https://archive.ics.uci.edu/ml/index.php. Jest to zbiór informacji na temat pożarów w parku Montensinho.

```
import pandas as pd
from sklearn import tree
fires_origin = pd.read_csv("forestfires.csv", header=0, index_col=False)
print(fires_origin.head())
```

```
   X  Y month  day  FFMC   DMC     DC  ISI  temp  RH  wind  rain  area
0  7  5   mar  fri  86.2  26.2   94.3  5.1   8.2  51   6.7   0.0   0.0
1  7  4   oct  tue  90.6  35.4  669.1  6.7  18.0  33   0.9   0.0   0.0
2  7  4   oct  sat  90.6  43.7  686.9  6.7  14.6  33   1.3   0.0   0.0
3  8  6   mar  fri  91.7  33.3   77.5  9.0   8.3  97   4.0   0.2   0.0
4  8  6   mar  sun  89.3  51.3  102.2  9.6  11.4  99   1.8   0.0   0.0
```

Obraz 1: Fragment kodu przedstawiający importowanie danych z pliku

Wśród danych znajdują się dane typu string, dlatego zostaną one na potrzeby ćwiczeń pominięte w docelowej bazie danych.

```
temp_months = fires_origin['month'].copy()
temp_days = fires_origin['day'].copy()
print(temp_months.head(), temp_days.head())
```

```
0    mar
1    oct
2    oct
3    mar
4    mar
Name: month, dtype: object 0    fri
1    tue
2    sat
3    fri
4    sun
Name: day, dtype: object
```

Obraz 2: Fragment kodu przedstawiający kolumny z danymi typu string, które zostały pomięte na potrzeby ćwiczenia

ii. Docelowa baza danych

```
d = {
    'X': fires_origin['X'].values,
    'Y': fires_origin['Y'].values,
    'temp': fires_origin['temp'].values,
    'RH': fires_origin['RH'].values,
    'wind': fires_origin['wind'].values,
    'rain': fires_origin['rain'].values,
    'area': fires_origin['area'].values
}
fires = pd.DataFrame(d)
print(fires.head())
```

```
   X  Y  temp  RH  wind  rain  area
0  7  5   8.2  51   6.7   0.0   0.0
1  7  4  18.0  33   0.9   0.0   0.0
2  7  4  14.6  33   1.3   0.0   0.0
3  8  6   8.3  97   4.0   0.2   0.0
4  8  6  11.4  99   1.8   0.0   0.0
```

Obraz 3: Podgląd pierwszych 5 wierszy docelowego zbioru danych

W celu przypisania potrzebnych klas, do których będą się odnosić testowane klasyfikatory, przeprowadzono prostą statystykę danych i przydzielono klasy „1” lub „0” tak, aby zbiór był niezbalansowany.

```
import statistics as stat
maximum = max(fires['temp'])
minimum = min(fires['temp'])
median = stat.median(fires['temp'])
diff = round((maximum - median)/4)
```

Obraz 4: Fragment kodu przedstawiający wartości zmiennych wykorzystanych w algorytmie klasyfikującym

```
# zbiór niezbalansowany prosty
tp = fires['temp']
serious = fires['temp'].copy()
for i, val in enumerate(fires['temp']):
    if (val >= (median + diff)):
        serious.loc[i] = 1
    else:
        serious.loc[i] = 0
```

Obraz 5: Fragment kodu przedstawiający algorytm klasyfikujący

Na koniec wszystkie kolumny ustawiono jako DataFrame:

```
x = pd.DataFrame(fires.values[:,0:13],
                 columns =['X','Y','temp', 'RH', 'wind', 'rain', 'area'])
y = pd.DataFrame(serious.values, columns =['SERIOUS'])
```

```
print(x.shape)
print(y.shape)
```

```
(517, 7)
(517, 1)
```

```
print(x[1:5])
print(y[1:5])
```

```
     X    Y  temp    RH  wind  rain  area
1  7.0  4.0  18.0  33.0   0.9   0.0   0.0
2  7.0  4.0  14.6  33.0   1.3   0.0   0.0
3  8.0  6.0   8.3  97.0   4.0   0.2   0.0
4  8.0  6.0  11.4  99.0   1.8   0.0   0.0
   SERIOUS
1      0.0
2      0.0
3      0.0
4      0.0
```

Obraz 6: Fragment kodu przedstawiający ostateczny układ zbioru danych wykorzystanego do ćwiczeń

3. Walidacja krzyżowa i stratyfikacja

3.1. Zadanie #1

i. Treść polecenia

Proszę pobrać inny zbiór danych, zmienić klasy w taki sposób żeby zbiór był niezbalansowany oraz przetestować drzewo decyzyjne stosując 5-krotną walidację krzyżową ze stratyfikacją

ii. Rozwiązanie

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.9, random_state=42)
```

Obraz 7: Fragment kodu z importem biblioteki train_test_split oraz dokonanie podziału klasyfikatorem binarnym na zbiór trenujący i testujący

```
train_class1 = 0;
train_class2 = 0;
for val in y_train['SERIOUS']:
    if (val == 0.0):
        train_class1 = train_class1 + 1
    else:
        if (val == 1.0):
            train_class2 = train_class2 + 1
test_class1 = 0;
test_class2 = 0;
for val in y_test['SERIOUS']:
    if (val == 0.0):
        test_class1 = test_class1 + 1
    else:
        if (val == 1.0):
            train_class2 = train_class2 + 1
print('0.0:', train_class1, '1.0:', train_class2)
print('0.0:', test_class1, '1.0:', test_class2)
# widzimy, że zbiór jest niezbalansowany
```

```
0.0: 39 1.0: 141
0.0: 337 1.0: 0
```

Obraz 8: Fragment kodu pokazujący, że wybrany zbiór danych jest niezbalansowany

```
from sklearn.model_selection import StratifiedKFold
skf = StratifiedKFold(n_splits=5, shuffle=True)
```

Obraz 9: Fragment kodu przedstawiający zastosowanie stratyfikacji

Wyniki dokładności modelu były takie same niezależnie od test_size.

```
from sklearn.model_selection import cross_val_score
clf = tree.DecisionTreeClassifier()
clf = clf.fit(x_train, y_train.values.ravel())
print(cross_val_score(clf, x, y, cv=skf))
```

```
[1. 1. 1. 1. 1.]
```

Obraz 10: Fragment kodu przedstawiający zastosowanie walidacji krzyżowej ze stratyfikacją; wyniki przedstawiają stopień dopasowania każdej iteracji krzyżowania i dla każdego test_size

Tabela 1: Wyniki dokładności modelu z klasyfikatorem binarnym ze stratyfikacją i walidacją krzyżową

	Test_size = 0.1	**Test_size = 0.5**	**Test_size = 0.9**
Klasyfikator binarny ze stratyfikacją i walidacją krzyżową	1.0	1.0	1.0

4. Klasyfikator drzewa losowego

4.1. Zadanie #2

i. Omówienie kodu

Zbiór danych z biblioteki iris zawierający przykładowe informacje na temat kwiatów, został podzielony równo na pół na zbiór trenujący i testujący. Następnie na zbiorach użyto klasyfikatora lasu losowego: najpierw z ustawieniami domyślnymi, potem z maksymalną ilością drzew równą dwa o maksymalnej głębokości równej 2.

```
# las losowy - zbior drzew decyzyjnych, np. 15
# 10 drzew: 0, 5 drzew: 1 -> klasa 0

# (I) budowanie drzew na innych podzbiorach danych
# (II) różne możliwe dostępne podzbiory cech na kolejnych poziomach
#  drzew decyzyjnych
# (III) polaczenie (I) i (II)

# 4 cechy: a,b,c,d

import pandas as pd
from sklearn.datasets import load_iris
from sklearn.ensemble import RandomForestClassifier
```

```
iris = load_iris()
x = pd.DataFrame(iris.data[:, :], columns = iris.feature_names[:])
y = pd.DataFrame(iris.target, columns =["Species"])
```

```
print(x.shape)
print(y.shape)
```

```
(150, 4)
(150, 1)
```

```
print(x[1:5])
print(y[1:5])
```

```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2
   Species
1        0
2        0
3        0
4        0
```

Obraz 11: Fragment kodu przedstawiający zaimportowanie danych z iris

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.5, random_state=42)
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(75, 4)
(75, 1)
(75, 4)
(75, 1)
```

Obraz 12: Fragment kodu pokazujący sposób dzielenia na zbiór trenujący i testujący - tutaj równo na pół

```
clf = RandomForestClassifier()
clf = clf.fit(x_train, y_train.values.ravel())
```

```
# Liczba drzew równa 2, maksymalna głębokość drzewa równa 2
```

```
clf_2_2 = RandomForestClassifier(n_estimators=2, max_depth=2)
clf_2_2 = clf_2_2.fit(x_train, y_train.values.ravel())
```

```
# Ewaluacja dwóch wariantów lasu drzew
```

```
y_test_pred_from_clf = clf.predict(x_test)
y_test_pred_from_clf_2_2 = clf_2_2.predict(x_test)
```

Obraz 13: Fragment kodu z zastosowaniem klasyfikatora lasu losowego

```
from sklearn.metrics import accuracy_score
```

```
print(accuracy_score(y_test, y_test_pred_from_clf))
print(accuracy_score(y_test, y_test_pred_from_clf_2_2))
```

```
0.9733333333333334
0.9333333333333333
```

Obraz 14: Fragment kodu z obliczeniami dokładności klasyfikatora lasu losowego

ii. Pytania

A. Czy uzyskane wyniki są zadowalające?

Dokładność dla obu przypadków użycia klasyfikatora jest ponad 90%, co jest bardzo dobrym wynikiem.

B. Czy model został przetestowany w prawidłowy sposób?

Nie, ponieważ istnieje ryzyko, że w zbiorze trenującym są same „0” i żadnych „1” albo w zbiorze testującym nie ma żadnych „1” więc nie zbadamy skuteczności modelu w dopasowywaniu „1”.

4.2. Zadanie #3

i. Treść polecenia

Proszę pobrać dowolny zbiór danych ze strony https://archive.ics.uci.edu/ml/index.php. Następnie proszę podzielić zbiór na dane trenujące i testujące, wytrenować 5 modeli lasów losowych z różną maksymalną głębokością i liczbą drzew w lesie, porównać wyniki. Proszę o sporządzenie sprawozdania z wnioskami.

ii. Rozwiązanie

A. Adnotacja odnośnie podziału na zbiór testujący i trenujący

Podczas wykonywania zadań, zauważono że proporcja zbioru testującego do trenującego podczas dokonywania podziału ma duże znaczenie, dlatego podczas zbierania wyników uwzględniono trzy różne sposoby podzielenia:

- testujący 10% i trenujący 90%: test_size = 0.1
- testujący 50% i trenujący 50%: test_size = 0.5
- testujący 90% i trenujący 10%: test_size = 0.9

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.1, random_state=42)
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(465, 7)
(465, 1)
(52, 7)
(52, 1)
```

Obraz 15: Fragment kodu przedstawiający liczność zbiorów po podziale dla test_size = 0.1

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.5, random_state=42)
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(258, 7)
(258, 1)
(259, 7)
(259, 1)
```

Obraz 16: Fragment kodu przedstawiający liczność zbiorów po podziale dla test_size = 0.5

```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.9, random_state=42)
```

```
print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(51, 7)
(51, 1)
(466, 7)
(466, 1)
```

Obraz 17: Fragment kodu przedstawiający liczność zbiorów po podziale dla test_size = 0.9

B. Budowanie lasów losowych

Zbudowane lasy:

- 2 drzewa o maksymalnej głębokości równej 2,
- 10 drzew o maksymalnej głębokości równej 2,
- 30 drzew o maksymalnej głębokości równej 2,
- 50 drzew o maksymalnej głębokości równej 2,
- 70 drzew o maksymalnej głębokości równej 2
- 10 drzew o maksymalnej głębokości równej 1,
- 10 drzew o maksymalnej głębokości równej 3.

```
from sklearn.ensemble import RandomForestClassifier
# domyślne wartości: 10 drzew i głębkość bez ograniczeń
clf = RandomForestClassifier()
clf = clf.fit(x_train, y_train.values.ravel())
```

```
# Liczba drzew równa 2, maksymalna głębokość drzewa równa 2
clf_2_2 = RandomForestClassifier(n_estimators=2, max_depth=2)
clf_2_2 = clf_2_2.fit(x_train, y_train.values.ravel())
```

```
# Liczba drzew równa 10, maksymalna głębokość drzewa równa 2
clf_10_2 = RandomForestClassifier(n_estimators=10, max_depth=2)
clf_10_2 = clf_10_2.fit(x_train, y_train.values.ravel())
```

```
# Liczba drzew równa 30, maksymalna głębokość drzewa równa 2
clf_30_2 = RandomForestClassifier(n_estimators=30, max_depth=2)
clf_30_2 = clf_30_2.fit(x_train, y_train.values.ravel())
```

```
# Liczba drzew równa 50, maksymalna głębokość drzewa równa 2
clf_50_2 = RandomForestClassifier(n_estimators=50, max_depth=2)
clf_50_2 = clf_50_2.fit(x_train, y_train.values.ravel())
```

```
# Liczba drzew równa 70, maksymalna głębokość drzewa równa 2
clf_70_2 = RandomForestClassifier(n_estimators=70, max_depth=2)
clf_70_2 = clf_70_2.fit(x_train, y_train.values.ravel())
```

```
# Liczba drzew równa 10, maksymalna głębokość drzewa równa 1
clf_10_1 = RandomForestClassifier(n_estimators=10, max_depth=1)
clf_10_1 = clf_10_1.fit(x_train, y_train.values.ravel())
```

```
# Liczba drzew równa 10, maksymalna głębokość drzewa równa 3
clf_10_3 = RandomForestClassifier(n_estimators=10, max_depth=3)
clf_10_3 = clf_10_3.fit(x_train, y_train.values.ravel())
```

Obraz 18: Fragment kodu przedstawiający sposób tworzenia lasów losowych o różnych parametrach

```
# ewaluacja różnych wariantów drzew
y_test_pred_from_clf = clf.predict(x_test)
y_test_pred_from_clf_2_2 = clf_2_2.predict(x_test)
y_test_pred_from_clf_10_2 = clf_10_2.predict(x_test)
y_test_pred_from_clf_30_2 = clf_30_2.predict(x_test)
y_test_pred_from_clf_50_2 = clf_50_2.predict(x_test)
y_test_pred_from_clf_70_2 = clf_70_2.predict(x_test)
y_test_pred_from_clf_10_1 = clf_10_1.predict(x_test)
y_test_pred_from_clf_10_3 = clf_10_3.predict(x_test)
```

Obraz 19: Fragment kodu przedstawiający ewaluację utworzonych lasów losowych

C. Porównanie wyników

```
Domyślne ustawienia: 1.0
Ilość drzew: 2, maksymalna głębokość: 2:   0.75
Ilość drzew: 10, maksymalna głębokość: 2:   0.9807692307692307
Ilość drzew: 30, maksymalna głębokość: 2:   1.0
Ilość drzew: 50, maksymalna głębokość: 2:   1.0
Ilość drzew: 70, maksymalna głębokość: 2:   1.0
Ilość drzew: 10, maksymalna głębokość: 1:   0.6538461538461539
Ilość drzew: 10, maksymalna głębokość: 3:   0.9615384615384616
```

Obraz 20: Fragment kodu przedstawiający wyniki dla test_size = 0.1

```
Domyślne ustawienia: 1.0
Ilość drzew: 2, maksymalna głębokość: 2:   0.7722007722007722
Ilość drzew: 10, maksymalna głębokość: 2:   0.9382239382239382
Ilość drzew: 30, maksymalna głębokość: 2:   0.9961389961389961
Ilość drzew: 50, maksymalna głębokość: 2:   0.9961389961389961
Ilość drzew: 70, maksymalna głębokość: 2:   0.9922779922779923
Ilość drzew: 10, maksymalna głębokość: 1:   0.7953667953667953
Ilość drzew: 10, maksymalna głębokość: 3:   0.9884169884169884
```

Obraz 21: Fragment kodu przedstawiający wyniki dla test_size = 0.5

```
Domyślne ustawienia: 0.9785407725321889
Ilość drzew: 2, maksymalna głębokość: 2:   0.9935622317596566
Ilość drzew: 10, maksymalna głębokość: 2:   0.9914163090128756
Ilość drzew: 30, maksymalna głębokość: 2:   0.9935622317596566
Ilość drzew: 50, maksymalna głębokość: 2:   0.9892703862660944
Ilość drzew: 70, maksymalna głębokość: 2:   0.9785407725321889
Ilość drzew: 10, maksymalna głębokość: 1:   0.8626609442060086
Ilość drzew: 10, maksymalna głębokość: 3:   0.9785407725321889
```

Obraz 22: Fragment kodu przedstawiający wyniki dla test_size = 0.9

Tabela 2: Wyniki dokładności modelu w zależności od ilości drzew
w lesie losowym oraz wielkości zbioru testowego przy **maksymalnej głębokości równej 2**

	Test_size = 0.1	**Test_size = 0.5**	**Test_size = 0.9**
Domyślnie (100 drzew i nieograniczona głębokość)	1.0	1.00	0.98
10 drzew	0.98	0.938	0.99
30 drzew	1.0	0.996	0.99
50 drzew	1.0	0.996	0.99
70 drzew	1.0	0.992	0.98

Tabela 3: Wyniki dokładności modelu w zależności od maksymalnej głębokości drzew w lesie losowym oraz wielkości zbioru testowego przy **ilości drzew w lasach losowych równej 10**

	Test_size = 0.1	**Test_size = 0.5**	**Test_size = 0.9**
Domyślnie (100 drzew i nieograniczona głębokość)	1.0	1.00	0.98
Głębokość = 1	0.65	0.80	0.86
Głębokość = 2	0.98	0.94	0.99
Głębokość = 3	0.96	0.99	0.98

4.3. Wnioski

Podczas korzystania z klasyfikatora lasu losowego duże znaczenie ma wielkość zbioru testującego i maksymalna głębokość drzew decydujących. Na podstawie zebranych wyników można wywnioskować, że najbardziej optymalne jest użycie jak największego zbioru testującego i jak największej głębokości drzew decyzyjnych. Warto również zauważyć, że domyślne ustawienia klasyfikatora lasu losowego otrzymały najlepsze wyniki.

5. Naiwny klasyfikator bayesowski

5.1. Zadanie #4

i. Treść polecenia

Proszę pobrać dowolny zbiór danych ze strony https://archive.ics.uci.edu/ml/index.php Następnie proszę podzielić zbiór na dane trenujące i testujące, wytrenować i przetestować klasyfikator GaussianNB. Proszę o sporządzenie sprawozdania z wnioskami.

ii. Rozwiązanie

Podczas dokonywania obliczeń dla naiwnego klasyfikatora bayesowski również przetestowano 3 podziały na zbiór testujący i trenujący, jak w przypadku pomiarów dla Zadanie #3.

```
from sklearn.naive_bayes import GaussianNB
clf = GaussianNB()
clf = clf.fit(x_train, y_train.values.ravel())
```

Obraz 23: Fragment kodu przedstawiający użycie naiwnego klasyfikatora bayesowskiego

```
from sklearn.metrics import accuracy_score
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_test, y_test_pred_from_clf))
```

```
0.7884615384615384
```

Obraz 24: Fragment kodu przedstawiający wynik dokładności modelu dla test_size - 0.1

```
from sklearn.metrics import accuracy_score
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_test, y_test_pred_from_clf))
```

```
0.7992277992277992
```

Obraz 25: Fragment kodu przedstawiający wynik dokładności modelu dla test_size - 0.5

```
from sklearn.metrics import accuracy_score
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_test, y_test_pred_from_clf))
```

```
0.9291845493562232
```

Obraz 26: Fragment kodu przedstawiający wynik dokładności modelu dla test_size - 0.9

Tabela 4: Wyniki dokładności modelu dla naiwnego klasyfikatora bayesowskiego w zależności od wielkości zbioru testującego

	Test_size = 0.1	**Test_size = 0.5**	**Test_size = 0.9**
Naiwny klasyfikator bayesowski	0.79	0.80	0.93

5.2. Wnioski

Naiwny klasyfikator bayesowski jest szczególnie czuły na wielkość zbioru testującego, najlepiej sobie radzi gdy zbiór trenujący jest bardzo mały.

6. Porównanie klasyfikatorów

Do porównania klasyfikatorów użyto najlepsze otrzymane wyniki dla poszczególnych klasyfikatorów.

Tabela 5: Zebranie wyników dokładności modelu dla wszystkich przetestowanych klasyfikatorów w zależności od wielkości zbioru testowego

	Test_size = 0.1	**Test_size = 0.5**	**Test_size = 0.9**
Klasyfikator binarny ze stratyfikacją i walidacją krzyżową	1.0	1.0	1.0
Klasyfikator lasu losowego (100 drzew i nieograniczona głębokość)	1.0	1.00	0.98
Naiwny klasyfikator bayesowski	0.79	0.80	0.93

7. Wnioski końcowe

Dla wybranego zbioru danych najlepszym rozwiązaniem okazał się klasyfikator binarny ze stratyfikacją i walidacją krzyżową, bo uzyskał dokładność 100% niezależnie od wielkości zbioru testowego. Najgorszy wynik otrzymał naiwny klasyfikator bayesowski. Warto jednak mieć na uwadze, że sposób przypisywania klas w algorytmie klasyfikującym był losowy, co mogło mieć duży wpływ na działanie tych klasyfikatorów.