Akademia Ekonomiczno-Humanistyczna w Warszawie

SPRAWOZDANIE

INTELIGENTNA ANALIZA DANYCH

LAB4

SIECI NEURONOWE I PORÓWNANIE KLASYFIKATORÓW

12.12.2021

JOANNA PRAJZENDANC

36358

MIŁOSZ SAKOWSKI

36381

Spis treści

1. Cel i przebieg ćwiczenia

Celem ćwiczenia było utrwalenie wiedzy w zakresie poznanych dotychczas klasyfikatorów: klasyfikator binarny, klasyfikator drzewa decyzyjnego, naiwny klasyfikator bayesowski, klasyfikator lasu losowego i klasyfikator sieci neuronowych. Podczas wykonywania zadań zastosowano również poznaną wcześniej stratyfikację, walidację krzyżową i optymalizację modelu grid search.

2. Definicje i założenia

2.1. Wyjaśnienie pojęć

W sprawozdaniu pojawiają się nowe pojęcia:

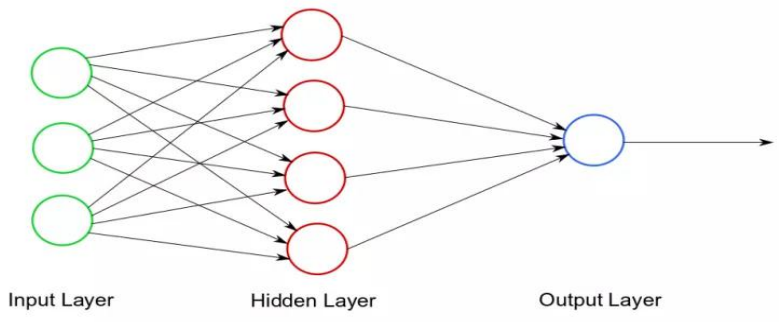
- ✧ sieci neuronowe - statystyczny model obliczeniowy stosowany w uczeniu maszynowym używany jako jeden z klasyfikatorów,
- ✧ neuron - warstwy sieci neuronowej, wyróżniamy 3 warstwy: warstwy wejścia (input layer), warstwy ukryte (hidden layer), oraz warstwy wyjścia (output layer).

3. Sieci neuronowe

3.1. Omówienie klasyfikatora

Sieć neuronowa to statystyczny model obliczeniowy stosowany w uczeniu maszynowym. Można o nim myśleć jak o systemie połączonych synapsami neuronów, które przesyłają między sobą impulsy (dane). Sieć neuronowa składa się z trzech warstw:

- ✧ warstwy wejścia (input layer),
- ✧ warstwy ukrytej (hidden layer),
- ✧ oraz warstwy wyjścia (output layer).



Obraz 1: Rysunek schematyczny sieci neuronowej z wyróżnionymi warstwami

Warstwa wejścia przyjmuje dane wejściowe do obliczeń, w warstwie ukrytej odbywają się wszystkie obliczenia. Wynik tych obliczeń jest przesyłany do warstwy wyjścia.

Na powyższym diagramie okręgi reprezentują neurony, zaś strzałki - synapsy. Każda synapsa ma przypisaną pewną wagę, tzn. liczbę, która (nieco upraszczając) określa, jak silnie przesyłana wartość wpływa na ostateczny wynik obliczeń. Żeby przesłać wartość,

synapsa najpierw czyta wartość z neuronu wejściowego, następnie wartość tę mnoży przez wagę, by w końcu przesłać wynik do neuronu wyjściowego. Następnie neuron wyjściowy dokonuje obliczeń na dostarczonych mu przez synapsy wartościach i otrzymany wynik przekazuje do wychodzącej z niego synapsy.

3.2. Zadanie #1

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 5 sieci neuronowych o różnych architekturach. Proszę o sporządzenie sprawozdania z wnioskami.

ii. Rozwiązanie

A. Przygotowanie danych

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.

Na potrzeby zadania pominięto kolumny z wartościami tekstowymi, ponieważ ich zamiana na wartości liczbowe była czasochłonna, a przydatność tych danych niewielka.

```
import pandas as pd
import numpy as np
import statistics as stat
from sklearn.model_selection import train_test_split
from sklearn.neural_network import MLPClassifier
```

Obraz 2: Użyte biblioteki

```
fires_origin = pd.read_csv(
    "forestfires.csv", header=0, index_col=False)
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: Wybrane kolumny z pliku forestfires.csv

```
maximum = max(fires['temp'])
minimum = min(fires['temp'])
median = stat.median(fires['temp'])
diff = round((maximum - median)/4)
max_RH = max(fires['RH'])
min_RH = min(fires['RH'])
median_RH = stat.median(fires['RH'])
max_wind= max(fires['wind'])
min_wind= min(fires['wind'])
median_wind= stat.median(fires['wind'])
```

```
tp = fires['temp']
rh = fires['RH']
w = fires['wind']
serious = fires['temp'].copy()
for i, val in enumerate(fires['temp']):
    #print(val, rh[i], serious[i])
    if (val >= (median + diff) and rh[i] <= 30 and w[i] >= 3.0):
        serious.loc[i] = 4
    else:
        if (val >= median + diff):
            serious.loc[i] = 3
        else:
            if (val >= (median - diff) and rh[i] <= 50 and w[i] >= 4.0):
                serious.loc[i] = 3
            else:
                if (val >= median - 2*diff and rh[i] <= 50 and w[i] >= 5.0):
                    serious.loc[i] = 2
                else:
                    if (val >= median - 2*diff):
                        serious.loc[i] = 1
                    else:
                        serious.loc[i] = 0
```

Obraz 4: Utworzenie kolumny z danymi do klasyfikacji na podstawie własnego algorytmu

```
x = np.array(fires.values)
y = np.array(serious.values)
```

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

```
[[ 7.   5.   8.2 51.   6.7  0.   0. ]
 [ 7.   4.  18.  33.   0.9  0.   0. ]
 [ 7.   4.  14.6 33.   1.3  0.   0. ]
 [ 8.   6.   8.3 97.   4.   0.2  0. ]
 [ 8.   6.  11.4 99.   1.8  0.   0. ]]
[0. 1. 1. 0. 0.]
```

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

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

Obraz 5: Podział danych na zbiór trenujący i testujący

B. Wyniki dokładności sieci neuronowych z różną ilością neuronów

- hidden_layer_sizes - parametr ten reprezentuje liczbę neuronów w warstwie ukrytej.
- random_state - określa generowanie liczb losowych na potrzeby inicjalizacji wag i odchyleń.
- max_iter - parametr ten określa liczbę epok, ile razy każdy punkt danych zostanie użyty.

W zadaniu zbudowano 5 różnych sieci neuronowych:

1. Ilość neuronów: 2,
2. Ilość neuronów: 5,
3. Ilość neuronów: 10,
4. Ilość neuronów: 30,
5. Ilość neuronów: 50.

```
clf_2 = MLPClassifier(hidden_layer_sizes=(2),
                      random_state=42,
                      max_iter=5000)
clf_5 = MLPClassifier(hidden_layer_sizes=(5),
                      random_state=42,
                      max_iter=5000)
clf_10 = MLPClassifier(hidden_layer_sizes=(10),
                       random_state=42,
                       max_iter=5000)
clf_30 = MLPClassifier(hidden_layer_sizes=(30),
                       random_state=42,
                       max_iter=5000)
clf_50 = MLPClassifier(hidden_layer_sizes=(50),
                       random_state=42,
                       max_iter=5000)
clf_2 = clf_2.fit(x_train, y_train)
clf_5 = clf_5.fit(x_train, y_train)
clf_10 = clf_10.fit(x_train, y_train)
clf_30 = clf_30.fit(x_train, y_train)
clf_50 = clf_50.fit(x_train, y_train)
```

Obraz 6: Zastosowanie klasyfikatora sieci neuronowych z różną ilością neuronów

```
y_test_pred_from_clf_2 = clf_2.predict(x_test)
y_test_pred_from_clf_5 = clf_5.predict(x_test)
y_test_pred_from_clf_10 = clf_10.predict(x_test)
y_test_pred_from_clf_30 = clf_30.predict(x_test)
y_test_pred_from_clf_50 = clf_50.predict(x_test)
```

```
from sklearn.metrics import accuracy_score
print('2: ', accuracy_score(y_test, y_test_pred_from_clf_2))
print('5: ', accuracy_score(y_test, y_test_pred_from_clf_5))
print('10: ', accuracy_score(y_test, y_test_pred_from_clf_10))
print('30: ', accuracy_score(y_test, y_test_pred_from_clf_30))
print('50: ', accuracy_score(y_test, y_test_pred_from_clf_50))
```

```
2:  0.3974358974358974
5:  0.7435897435897436
10:  0.8397435897435898
30:  0.8141025641025641
50:  0.8333333333333334
```

Obraz 7: Obliczenie dokładności modelu w zależności od ilości neuronów

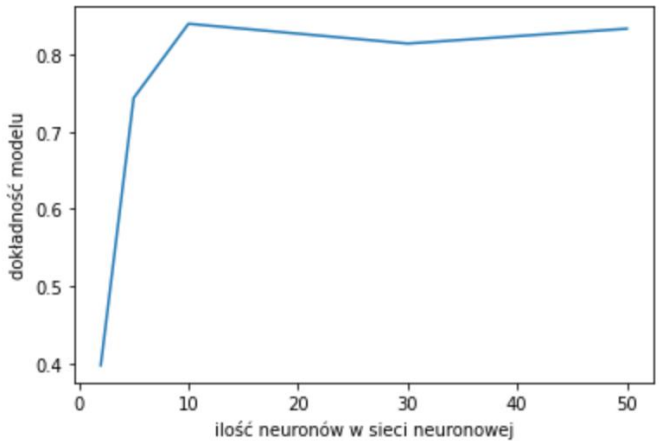
iii. Porównanie wyników i wnioski

Tabela 1: Zestawienie wyników dokładności modelu sieci neuronowej w zależności od liczby neuronów

Ilość neuronów	**Dokładność**
2	0,397
5	0,744
10	0,840
30	0,814
50	0,833

Dokładność modelu wyraźnie rośnie w zależności od liczby neuronów w sieci neuronowej.

```
import matplotlib.pyplot as plt
plt.plot([2,5,10,30,50],
         [accuracy_score(y_test, y_test_pred_from_clf_2),
          accuracy_score(y_test, y_test_pred_from_clf_5),
          accuracy_score(y_test, y_test_pred_from_clf_10),
          accuracy_score(y_test, y_test_pred_from_clf_30),
          accuracy_score(y_test, y_test_pred_from_clf_50)
         ])
plt.ylabel('dokładność modelu')
plt.xlabel('ilość neuronów w sieci neuronowej')
plt.show()
```



Obraz 8: Wykres zależności dokładności modelu od ilości neuronów

Prawdopodobnie zależność ta jest funkcją logarytmiczną, co oznacza że wartość dokładności szybko rośnie już dla niewielkiej liczby neuronów - dzięki czemu szybko możemy otrzymać zadowalające przybliżenie, ale też uzyskanie bardzo dużej dokładności (powyżej 90%) będzie wymagało sieci neuronów o bardzo dużej ilości neuronów, co przekłada się na czas obliczeń i duże ryzyko przetrenowania modelu.

4. Porównanie klasyfikatorów

4.1. Zadanie #2

i. Treść polecenia

Pobrać wybraną bazę danych i porównać wytrenowanie różnymi klasyfikatorami.

ii. Baza danych

A. Informacje techniczne

- ✧ data dodania bazy danych: 22.07.2019,
- ✧ charakterystyka: wielowymiarowa,
- ✧ kategoria: komputery,
- ✧ ilość danych: 125,
- ✧ brakujące wartości: brak.

Bazę danych pobrano z
https://archive.ics.uci.edu/ml/datasets/Alcohol+QCM+Sensor+Dataset

W pobranym zestawie danych znajdowało się 5 plików z wynikami pomiarów:

- QCM3.csv,
- QCM6.csv,
- QCM7.csv,
- QCM10.csv,
- QCM12.csv.

Nazwy plików odpowiadają numerowi czujnika QCM użytego podczas pomiarów.

B. Opis badania i zawartość zestawu danych[1]

Celem badania było zmierzenie reakcji różnych czujników QCM na 5 wybranych alkoholi, aby określić który z tych czujników będzie najlepszy do klasyfikacji tych alkoholi.

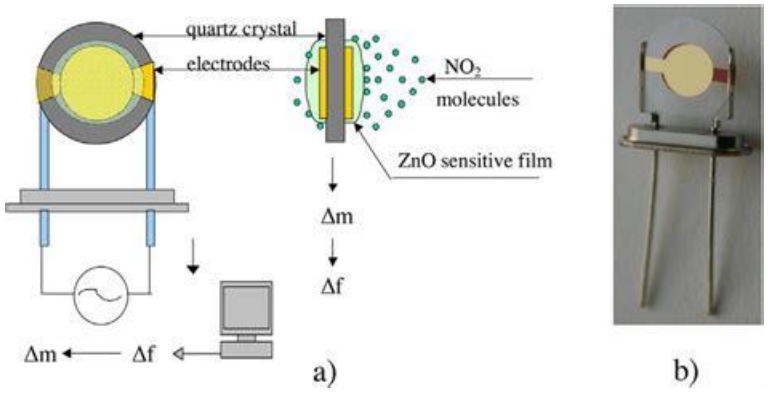
Pomiary dotyczyły 5 różnych gazów (alkoholi):

- 1-octanolu,
- 1-propanolu,
- 2-butanolu,
- 2-propanolu,
- 1-isobutanolu.

---

[1] Opracowanie na podstawie analizy tongahancepel: https://www.kaggle.com/tolgahancepel/qcm-sensor-alcohol-classification-using-keras/notebook
oraz artykułu „*Classification of alcohols obtained by QCM sensors with different characteristics using ABC based neural network*" M. Fatih Adak, Peter Lieberzeit, Purim Jarujamrus, Nejat Yumusak (https://www.sciencedirect.com/science/article/pii/S2215098619303337)

Wymienione gazy zostały zbadane przez 5 różnych sensorów QCM[2], czyli przez mikrowagi kwarcowe, która są rodzajem czujnika do wykrywania bardzo małych zmian masy. Mikrowaga kwarcowa działa na zasadzie rezonatora kwarcowego pracującego z drganiami ścinającymi[3] i jest używana do budowania tzw. *Elektronicznego nosa*[4].



Obraz 9: a) rysunek schematyczny mikrowagi kwarcowej; b) zdjęcie przykładowej mikrowagi kwarcowej

Rezonator kwarcowy składa się z dwóch okręgów, które różnią się zawartością MIP[5] i NP[6]. Każdy z okręgów to osobny kanał pomiaru drgań (kanał pomiaru).

Tabela 2: Stosunki MIP i NP w każdym z czujników

Czujnik	**MIP**	**NP**
QCM3	1	1
QCM6	1	0
QCM7	1	0,5
QCM10	1	2
QCM12	0	1

Podłączając mikrowagę do układu elektronicznego, można zmierzyć zmianę częstotliwości drgań rezonatora, która odpowiada zmianie masy. Dzięki temu można „zważyć” gaz.

---

[2] the Qaurtz Crystal Microbalance - mikrowaga kwarcowa

[3] Informacje pochodzą z Wikipedii: https://pl.wikipedia.org/wiki/Mikrowaga_kwarcowa

[4] Więcej na Wikipedii: https://pl.wikipedia.org/wiki/Elektroniczny_nos

[5] moleculary imprinted polymers -

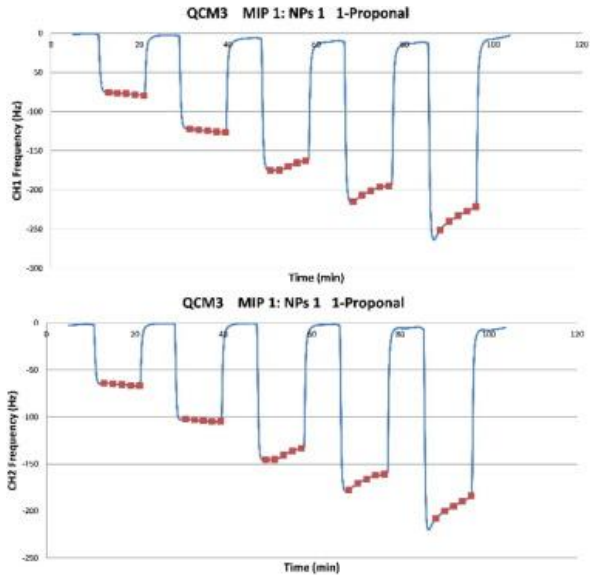
[6] Nanoparticles -

Jeden pomiar dla jednego czujnika trwał 120 min, w trakcie tego czasu najpierw czujnik był umieszczany na 30 min w czystym powietrzu w celu oczyszczenia, następnie wybrany gaz był dodawany do powietrza aż do uzyskania zadanej koncentracji powietrze - alkohol i dokonywano pomiaru. Przed kolejnym pomiarem w innej koncentracji tego samego gazu czujnik był oczyszczany przez 7 min w czystym powietrzu.

Tabela 3: Wartości koncentracji powietrze - alkohol, dla których wykonano pomiary

L.p.	Koncentracja powietrza	Koncentracja alkoholu
1	0,799	0,201
2	0,700	0,300
3	0,600	0,400
4	0,501	0,499
5	0,400	0,600

Wszystkie pomiary zostały przeprowadzone w temperaturze pokojowej 25°C. Alkohol w stanie płynnym był przelewany do szklanej tuby o pojemności 50ml z umieszczonym czujnikiem. Próbka alkoholu docierała do czujnika jako gaz i wyniki zmiany częstotliwości drgań w Hz z każdego z dwóch kanałów mikrowagi kwarcowej były przesyłane do komputera.



Obraz 10: Przykład pomiarów zmiany częstotliwości dla 1-Proponalu z czujnika QCM3

C. Przygotowanie do klasyfikacji

```
import pandas as pd
import numpy as np
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import StratifiedKFold
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import cross_val_score
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn import tree
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier
```

Obraz 11: Użyte biblioteki

```
qcm3 = pd.read_csv('Dataset/QCM3.csv', sep = ';')
qcm6 = pd.read_csv('Dataset/QCM6.csv', sep = ';')
qcm7 = pd.read_csv('Dataset/QCM7.csv', sep = ';')
qcm10 = pd.read_csv('Dataset/QCM10.csv', sep = ';')
qcm12 = pd.read_csv('Dataset/QCM12.csv', sep = ';')
```

```
print(qcm3.head())
```

```
   0.799_0.201  0.799_0.201.1  0.700_0.300  0.700_0.300.1  0.600_0.400  \
0       -10.06         -10.62       -14.43         -18.31       -24.64
1        -9.69         -10.86       -16.73         -21.75       -28.47
2       -12.07         -14.28       -21.54         -27.92       -35.19
3       -14.21         -17.41       -25.91         -33.36       -41.29
4       -16.57         -20.35       -29.97         -37.84       -47.03

   0.600_0.400.1  0.501_0.499  0.501_0.499.1  0.400_0.600  0.400_0.600.1  \
0         -30.56       -38.62         -45.59       -54.89         -62.28
1         -35.83       -43.65         -52.43       -61.92         -71.27
2         -43.94       -52.04         -62.49       -71.97         -83.10
3         -51.27       -59.94         -71.55       -81.51         -93.83
4         -57.29       -67.13         -78.96       -90.01        -102.65

   1-Octanol  1-Propanol  2-Butanol  2-propanol  1-isobutanol
0          1           0          0           0             0
1          1           0          0           0             0
2          1           0          0           0             0
3          1           0          0           0             0
4          1           0          0           0             0
```

Obraz 12: Załadowanie danych z plików

```
dataset = pd.concat([qcm3, qcm6, qcm7, qcm10, qcm12])
print("Shape of dataset: ", dataset.shape)
```

```
Shape of dataset:  (125, 15)
```

Obraz 13: Połączenie danych w jeden zestaw

```
# dodanie kolumny z wartościami liczbowymi dla każdego rodzaju alkoholu
dataset.loc[dataset["1-Octanol"] == 1, 'alcohol'] = 1
dataset.loc[dataset["1-Propanol"] == 1, 'alcohol'] = 2
dataset.loc[dataset["2-Butanol"] == 1, 'alcohol'] = 3
dataset.loc[dataset["2-propanol"] == 1, 'alcohol'] = 4
dataset.loc[dataset["1-isobutanol"] == 1, 'alcohol'] = 5
dataset['alcohol'].value_counts()
```

```
1.0    25
2.0    25
3.0    25
4.0    25
5.0    25
Name: alcohol, dtype: int64
```

Obraz 14: Dodanie kolumny sumarycznej klasyfikującej rodzaj użytego alkoholu

iii. Klasyfikator: binarny

```
######################
# klasyfikator binarny
######################
```

```
x = np.array(dataset.values[:,:10])
y = np.array(dataset.values[:,15])
```

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

```
[[ -10.06  -10.62  -14.43  -18.31  -24.64  -30.56  -38.62  -45.59  -54.89
   -62.28]
 [  -9.69  -10.86  -16.73  -21.75  -28.47  -35.83  -43.65  -52.43  -61.92
   -71.27]
 [ -12.07  -14.28  -21.54  -27.92  -35.19  -43.94  -52.04  -62.49  -71.97
   -83.1 ]
 [ -14.21  -17.41  -25.91  -33.36  -41.29  -51.27  -59.94  -71.55  -81.51
   -93.83]
 [ -16.57  -20.35  -29.97  -37.84  -47.03  -57.29  -67.13  -78.96  -90.01
  -102.65]]
[1. 1. 1. 1. 1.]
```

```
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)
```

```
(12, 10)
(12,)
(113, 10)
(113,)
```

Obraz 15: Podział na zbiór trenujący i testujący

```
# stratyfikacja
skf = StratifiedKFold(n_splits=5, shuffle=True)
```

```
# walidacja krzyżowa
# skalowanie maksymalnej iteracji
model = make_pipeline(StandardScaler(), LogisticRegression())
model.fit(x_train, y_train)
# pomiar dokładności bez stratyfikacji i walidacji krzyżowej
y_test_pred_from_model = model.predict(x_test)
print(accuracy_score(y_test, y_test_pred_from_model))
# pomiar dokładności ze stratyfikacją i walidacją krzyżową
print(cross_val_score(model, x, y, cv=skf))
```

```
0.4690265486725664
[0.76 0.52 0.56 0.56 0.52]
```

Obraz 16: Obliczenie dokładności modelu dla klasyfikatora binarnego

Pomimo faktu, że zbiór jest zbalansowany stratyfikacja w połączeniu z walidacją krzyżową wydają się podnosić dokładność modelu.

Nie zastosowano optymalizacji modelu ze względu na złożoność klasyfikatora.

iv. Klasyfikator: drzewo decyzyjne

```
################################
# klasyfikator drzewa decyzyjnego
################################
```

```
x = np.array(dataset.values[:,:10])
y = np.array(dataset.values[:,15])
```

```
x_train, x_test, y_train, y_test = train_test_split(
    x, y, test_size=0.9, random_state=42)
```

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

```
[[ -10.06  -10.62  -14.43  -18.31  -24.64  -30.56  -38.62  -45.59  -54.89
   -62.28]
 [  -9.69  -10.86  -16.73  -21.75  -28.47  -35.83  -43.65  -52.43  -61.92
   -71.27]
 [ -12.07  -14.28  -21.54  -27.92  -35.19  -43.94  -52.04  -62.49  -71.97
   -83.1 ]
 [ -14.21  -17.41  -25.91  -33.36  -41.29  -51.27  -59.94  -71.55  -81.51
   -93.83]
 [ -16.57  -20.35  -29.97  -37.84  -47.03  -57.29  -67.13  -78.96  -90.01
  -102.65]]
[1. 1. 1. 1. 1.]
```

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

```
(12, 10)
(12,)
(113, 10)
(113,)
```

Obraz 17: Podział na zbiór trenujący i testujący

```
# stratyfikacja plus walidacja krzyżowa
clf = tree.DecisionTreeClassifier()
skf = StratifiedKFold(n_splits=5, shuffle=True)
clf = clf.fit(x_train, y_train)
# dokładność
y_test_pred_from_clf = clf.predict(x_test)
print('bez walidacji krzyżowej i stratyfikacji',
      accuracy_score(y_test, y_test_pred_from_clf))
print('walidacja i stratyfikacja',
      cross_val_score(clf, x, y, cv=skf))
```

```
bez walidacji krzyżowej i stratyfikacji 0.39823008849557523
walidacja i stratyfikacja [0.76 0.96 0.8  0.96 0.84]
```

Obraz 18: Obliczenie dokładności modelu z klasyfikatorem drzewa decyzyjnego bez optymalizacji

```
# optymalizacja modelu
criterion = np.array(['gini','entropy'])
max_depth = np.array([
    1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20])
grid = GridSearchCV(estimator = clf,
                    param_grid = dict(
                        max_depth=max_depth,
                        criterion=criterion))
grid.fit(x, y)
print(grid.best_params_)
```

```
{'criterion': 'gini', 'max_depth': 16}
```

```
clf = tree.DecisionTreeClassifier(
    criterion = grid.best_params_['criterion'],
    max_depth = grid.best_params_['max_depth'])
clf = clf.fit(x_train, y_train)
# dokładność
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_test, y_test_pred_from_clf))
print(cross_val_score(clf, x, y, cv=skf))
```

```
0.4247787610619469
[0.92 0.96 0.96 0.84 0.92]
```

Obraz 19: Optymalizacja modelu i obliczenie dokładności zoptymalizowanego modelu z klasyfikatorem drzewa decyzyjnego

Pomimo faktu, że zbiór jest zbalansowany stratyfikacja w połączeniu z walidacją krzyżową wydają się podnosić dokładność modelu.

v. Klasyfikator: las losowy

```
############################
# klasyfikator lasu losowego
############################
```

```
x = np.array(dataset.values[:,:10])
y = np.array(dataset.values[:,15])
```

```
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)
```

```
(12, 10)
(12,)
(113, 10)
(113,)
```

Obraz 20: Podział na zbiór trenujący i testujący

```
clf = RandomForestClassifier(n_estimators=2)
skf = StratifiedKFold(n_splits=5, shuffle=True)
clf = clf.fit(x_train, y_train)
# dokładność
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_test, y_test_pred_from_clf))
print(cross_val_score(clf, x, y, cv=skf))
```

```
0.34513274336283184
[0.8  0.84 0.72 0.76 0.96]
```

Obraz 21: Obliczenie dokładności modelu z klasyfikatorem lasu losowego bez optymalizacji

```
# optymalizacja modelu
criterion = np.array(['gini','entropy'])
max_depth = np.array(
    [1,2,3,4,5,6,7,8,9,10,11,12,13,14,15])
n_estimators = np.array([2,3,4,5,6,7,8,9,10])
clf = RandomForestClassifier()
grid = GridSearchCV(estimator = clf,
                    param_grid = dict(
                        n_estimators = n_estimators,
                        max_depth=max_depth,
                        criterion=criterion))
grid.fit(x, y)
print(grid.best_params_)
```

```
{'criterion': 'entropy', 'max_depth': 12, 'n_estimators': 5}
```

```
clf = RandomForestClassifier(
    n_estimators = grid.best_params_['n_estimators'],
    criterion = grid.best_params_['criterion'],
    max_depth = grid.best_params_['max_depth'])
clf = clf.fit(x_train, y_train)
# dokładność
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_test, y_test_pred_from_clf))
print(cross_val_score(clf, x, y, cv=skf))
```

```
0.5132743362831859
[1.   0.92 0.84 0.92 0.96]
```

Obraz 22: Optymalizacja modelu i obliczenie dokładności zoptymalizowanego modelu z klasyfikatorem lasu losowego

Pomimo faktu, że zbiór jest zbalansowany stratyfikacja w połączeniu z walidacją krzyżową podnoszą dokładność modelu.

vi. Klasyfikator: naiwny bayesowski

```
################################
# naiwny klasyfikator bayesowski
################################
```

```
x = np.array(dataset.values[:,:10])
y = np.array(dataset.values[:,15])
```

```
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)
```

```
(12, 10)
(12,)
(113, 10)
(113,)
```

Obraz 23: Podział na zbiór trenujący i testujący

```
clf = GaussianNB()
clf = clf.fit(x_train, y_train)
```

```
# dokładność
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_test, y_test_pred_from_clf))
skf = StratifiedKFold(n_splits=5, shuffle=True)
print(cross_val_score(clf, x, y, cv=skf))
```

```
0.2743362831858407
[0.52 0.48 0.32 0.4  0.56]
```

Obraz 24: Obliczenie dokładności modelu z naiwnym klasyfikatorem bayesowskim bez optymalizacji

Pomimo faktu, że zbiór jest zbalansowany stratyfikacja w połączeniu z walidacją krzyżową wydają się podnosić dokładność modelu.

Nie zastosowano optymalizacji modelu ze względu na złożoność klasyfikatora.

vii. Klasyfikator: sieci neuronowe

```
################################
# klasyfikator sieci neuronowych
################################
```

```
x = np.array(dataset.values[:,:10])
y = np.array(dataset.values[:,15])
```

```
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)
```

```
(12, 10)
(12,)
(113, 10)
(113,)
```

Obraz 25: Podział na zbiór trenujący i testujący

```
# dokładność
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_test, y_test_pred_from_clf))
skf = StratifiedKFold(n_splits=5, shuffle=True)
print(cross_val_score(clf, x, y, cv=skf))
```

```
0.19469026548672566
[0.12 0.24 0.2  0.16 0.16]
```

Obraz 26: Obliczenie dokładności modelu z klasyfikatorem sieci neuronowych bez optymalizacji

```
# optymalizacja modelu
hidden_layer_sizes = np.array([5,10,30,50,100])
random_state = np.array([0,10,20,40,60])
max_iter = np.array([8000])
clf = MLPClassifier()
grid = GridSearchCV(estimator = clf,
                    param_grid = dict(
                        hidden_layer_sizes=(hidden_layer_sizes),
                        random_state=random_state,
                        max_iter=max_iter))
grid.fit(x, y)
print(grid.best_params_)
```

```
{'hidden_layer_sizes': 30, 'max_iter': 8000, 'random_state': 20}
```

```
clf = MLPClassifier(
    hidden_layer_sizes = (grid.best_params_['hidden_layer_sizes']),
    random_state = grid.best_params_['random_state'],
    max_iter = grid.best_params_['max_iter'])
clf = clf.fit(x_train, y_train)
# dokładność
y_test_pred_from_clf = clf.predict(x_test)
print(accuracy_score(y_test, y_test_pred_from_clf))
print(cross_val_score(clf, x, y, cv=skf))
```

```
0.5398230088495575
[0.92 0.96 0.88 0.88 0.8 ]
```

Obraz 27: Optymalizacja modelu i obliczenie dokładności zoptymalizowanego modelu z klasyfikatorem sieci neuronowych

Pomimo faktu, że zbiór jest zbalansowany stratyfikacja w połączeniu z walidacją krzyżową podnosi dokładność modelu.

4.2. Podsumowanie i wnioski

W porównaniu dokładności modeli wzięto wyniki dla najlepiej dopasowanego modelu - z zastosowaniem grid_search jeżeli było to możliwe oraz ze stratyfikacją i walidacją krzyżową, ponieważ pomiary wykazały, że dokładność każdego z modeli była wtedy większa.

Tabela 4: Zestawienie średnich dokładności najlepiej dopasowanych modeli dla różnych klasyfikatorów

Klasyfikator	**Średnia dokładność modelu**
binarny	0,560
drzewa decyzyjnego	0,896
lasu losowego	0,912
naiwny bayesowski	0,424
sieci neuronowe	0,920

Dla wybranego zestawu danych najlepiej poradziły sobie modele z klasyfikatorami: lasu losowego i sieci neuronowych, model z klasyfikatorem drzewa decyzyjnego również ma bardzo wysoką dokładność. Modele z klasyfikatorami: binarnym i naiwnym bayesowskim mają dokładność rzędu 50%, jednak warto zauważyć że podczas wykonywania ćwiczenia były to modele bez zastosowania optymalizacji grid_search.

Oprócz wyniku dokładności warto również pamiętać o złożoności klasyfikatorów i czasu wykonywania obliczeń przez program. Klasyfikator sieci neuronowych jest najbardziej złożonym z wszystkich użytych w ćwiczeniu, najtrudniej było wybrać zakres danych podczas optymalizacji modelu i czas wykonywania obliczeń był zdecydowanie najdłuższy. Klasyfikatory: drzewa decyzyjnego i lasu losowego mają równie wysoką dokładność, a ich złożoność jest dużo mniejsza i wykonują się szybciej.