Klasyfikator wspomagający diagnozę raka (Breast Cancer Wisconsin)

KWD

Michał Mamla

Dominik Pepaś

01.02.2021

Streszczenie

Wyżej wymieniony projekt ma na celu stworzenie klasyfikatora realizującego przewidywanie diagnozy dla pacjenta kliniki onkologicznej. Program komputerowy na podstawie wcześniej opracowanych danych, utworzonych na przykład w wywiadzie prowadzonym przez lekarza, trenuje klasyfikator który w kolejnym kroku zostaje poddany testom. Zadaniem naszym, czyli programistów jest analiza wyników programu i wybranie oraz opracowanie najbardziej wiarygodnego modelu klasyfikującego.

Spis treści

1 Wprowadzenie **1**
1.1 Opis problemu 1

2 Realizacja **2**
2.1 Wprowadzenie teoretyczne 2
2.2 Opis danych wejściowych 3
2.3 Normalizacja i standaryzacja danych 8
2.4 Analiza jakości uzyskanego modelu 8
2.5 Badania symulacyjne 10

3 Podsumowanie **17**

A Kod programu **18**

Rozdział 1

Wprowadzenie

Nowotwór piersi, czyli rak jest obecnie drugą główną przyczyną zgonów na świecie. W naszym kraju co roku z powodu zachorowań na złośliwe nowotwory piersi umiera około 6 tysięcy osób, na świecie ponad 600 tysięcy. Eksperci alarmują, że stale zwiększa się liczba pacjentów onkologicznych. Wczesne wykrycie zmian nowotworowych w ciele pacjenta może znacząco zwiększyć jego prognozy, czyli szanse na wyleczenie. Natomiast zidentyfikowanie zmiany jako łagodnej może uchronić pacjenta przed podjęciem niepotrzebnego, wyczerpującego leczenia. Znalezienie metody wykrywania zmian nowotworowych i klasyfikowania ich na różne kategorie może znacząco przyspieszyć cały proces leczenia, dlatego jest to aktualnie bardzo popularny oraz pożądany temat. Narzędziem, które sprawdza się w tego typu zadaniach jest uczenie maszynowe. Jego metody klasyfikowania danych są często wykorzystywane w medycynie, do diagnozowania przypadków i podejmowania decyzji.

1.1 Opis problemu

Podstawowym problemem jest stworzenie klasyfikatora, który przy klasyfikowaniu danych, czyli w tym przypadku cech pacjenta klinikii onkologicznej, będzie nieomylny lub też będzie popełniał możliwie najmniejszą ilość błędów. Pod uwagę należy wziąć odpowiedni dobór danych na których klasyfikator nas będzie trenowany. Na ich podstawie następnie będzie podejmować decyzję. Model musi także zostać bardzo dokładnie przetestowany, uwzględniając zmianę danych testowych i trenujących, tak by wykluczyć jak największą ilość pomyłek. Uwagę należy poświęcić także rozkładzie danych na treningowe i testujące. Tworząc model dbamy o to, by nie był przeuczony.

Rozdział 2

Realizacja

2.1 Wprowadzenie teoretyczne

Odmianą analizy regresji używaną do rozwiązania naszego problemu jest regresja logistyczna. Najważniejszą cechą regresji logistycznej jest to, że etykieta przyjmuje tylko dwie wartości (najczęściej 0 lub 1). Używamy jej do określania wystąpienia jakiegoś zdarzenia lub zjawiska. Regresja logistyczna różni się od regresji liniowej. W przypadku regresji logistycznej w przeciwieństwie do regresji liniowej celem nie jest przewidzenie wartości zmiennej zależnej na podstawie wykorzystania predykatów, ale przewidzenie prawdopodobieństwa na wystąpienie jakiegoś zdarzenia.

Wzór na równanie regresji logistycznej:

$$P(Y = 1|\, x_1, x_2, \dots, x_k) = P(X) = \frac{e^{a_0 + \sum_{i=1}^{k} a_i x_i}}{1 + e^{a_0 + \sum_{i=1}^{k} a_i x_i}}$$

Gdzie:

$a_i, i = 0, \dots, k$ są współczynnikami regresji,

$x_1, x_2, \dots, x_k$ - zmienne niezależne (ilościowe lub jakościowe).

Analiza regresji logistycznej w porównaniu do regresji liniowej obarczona jest mniejszą ilością założeń. Najważniejszą charakterystyką regresji logistycznej jest to, że predykatem jest zmienna dychotomiczna 0 - 1.

Metodą używaną podczas testowania naszego modelu jest walidacja krzyżowa. Jest to metoda statystyczna polegająca na podziale próby statystycznej na podzbiory, a następnie przeprowadzeniu wszelkich analiz na zbiorze uczącym, podczas gdy zbiór testowy służy do potwierdzenia wiarygodności wyników. Metoda ta jest stosowana w przypadku niedostatecznie licznego zbioru uczącego. Powtarzając proces kroswalidacji k razy i obliczając statystyki na zbiorach walidacyjnych tak zbudowanych modeli możliwa jest estymacja błędu i generalizacji modelu.

2.2 Opis danych wejściowych

Zbiór danych na których działać będzie klasyfikator składa się z 569 rekordów, z których każdy określa oddzielnego pacjenta klinikii. W każdej linii znajduje się 30 cech określających konkretnego pacjenta. Na ich podstawie wyznaczana jest etykieta, która określa rodzaj zmiany nowotworowej - złośliwej (0), bądź łagodnej (1).

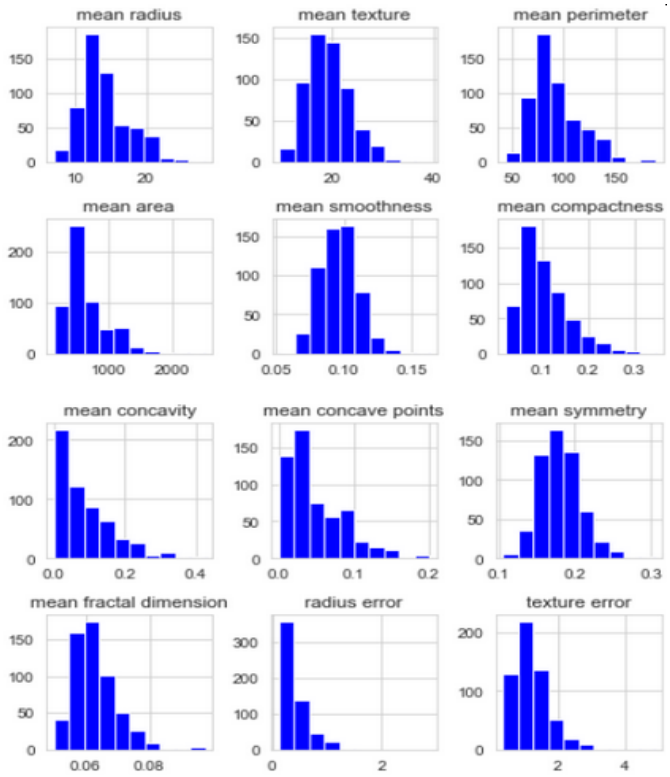
Specyfikacja określająca zbiór

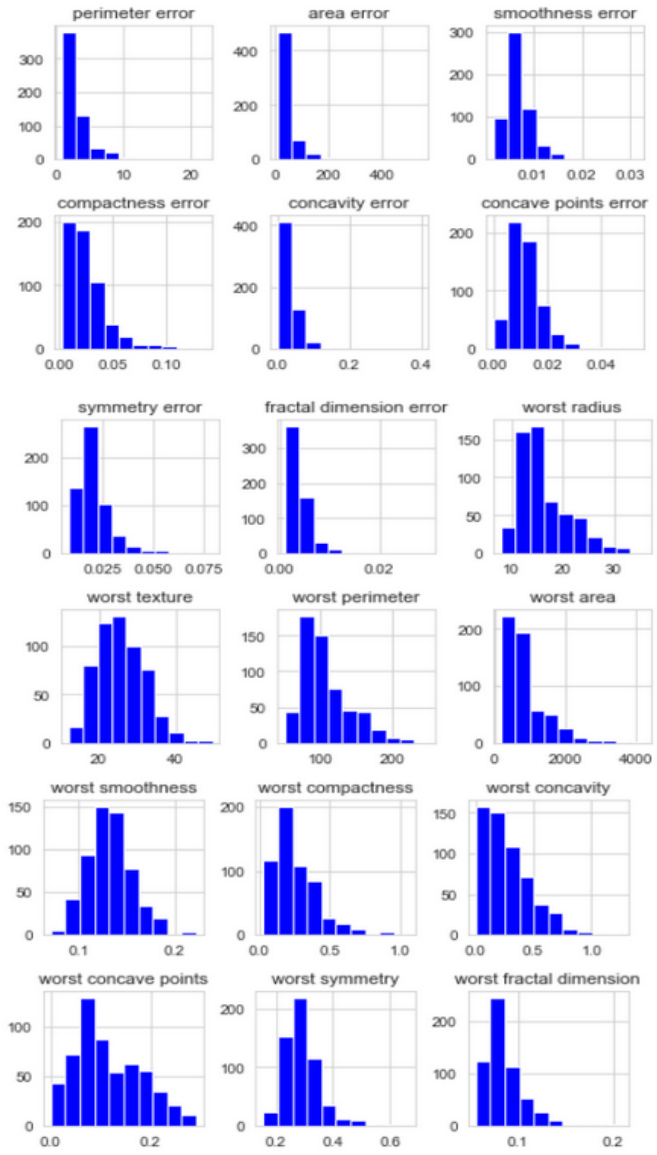
Classes	2
Samples per class	212(M),357(B)
Samples total	569
Dimensionality	30
Features	real, positive

Dane określające pacjenta:

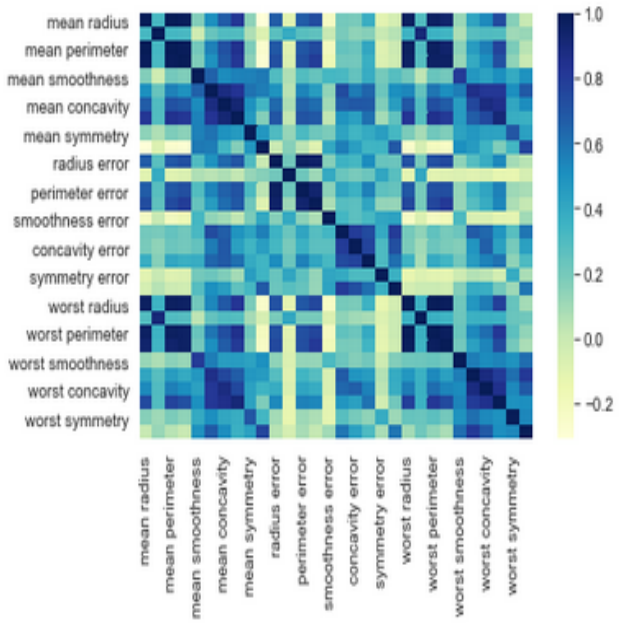
	Min	Max
radius (mean):	6.981	28.11
texture (mean):	9.71	39.28
perimeter (mean):	43.79	188.5
area (mean):	143.5	2501.0
smoothness (mean):	0.053	0.163
compactness (mean):	0.019	0.345
concavity (mean):	0.0	0.427
concave points (mean):	0.0	0.201
symmetry (mean):	0.106	0.304
fractal dimension (mean):	0.05	0.097
radius (standard error):	0.112	2.873
texture (standard error):	0.36	4.885
perimeter (standard error):	0.757	21.98
area (standard error):	6.802	542.2
smoothness (standard error):	0.002	0.031
compactness (standard error):	0.002	0.135
concavity (standard error):	0.0	0.396
concave points (standard error):	0.0	0.053
symmetry (standard error):	0.008	0.079
fractal dimension (standard error):	0.001	0.03
radius (worst):	7.93	36.04
texture (worst):	12.02	49.54
perimeter (worst):	50.41	251.2
area (worst):	185.2	4254.0
smoothness (worst):	0.071	0.223
compactness (worst):	0.027	1.058
concavity (worst):	0.0	1.252
concave points (worst):	0.0	0.291
symmetry (worst):	0.156	0.664
fractal dimension (worst):	0.055	0.208

Rozkład danych z podziałem na cechy pacjentów

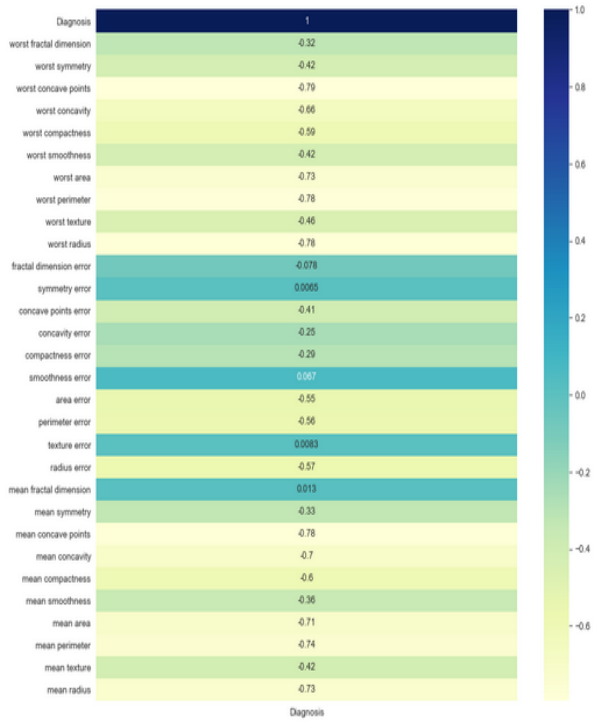



perimeter error
area error
smoothness error
compactness error
concavity error
concave points error
symmetry error
fractal dimension error
worst radius
worst texture
worst perimeter
worst area
worst smoothness
worst compactness
worst concavity
worst concave points
worst symmetry
worst fractal dimension

Wzajemna korelacja cech



Wzajemna korelacja cech z diagnozą



2.3 Normalizacja i standaryzacja danych

Biorąc pod uwagę zróżnicowanie zakresu danych w zbiorze, w obliczeniach konieczne jest uwzględnienie normalizacji oraz standaryzacji. Wiele klasyfikatorów oblicza odległość między dwoma punktami na podstawie odległości euklidesowej. Jeśli jedna z cech ma szeroki zakres wartości, odległość będzie zależała od tej konkretnej cechy. Dlatego zakres wszystkich cech powinien zostać znormalizowany, tak aby każda cecha miała udział w przybliżeniu proporcjonalnie do końcowej odległości. Innym powodem stosowania skalowania cech jest to, że zejście gradientu zbiega się znacznie szybciej ze skalowaniem cech niż bez niego. Standaryzacja odnosi się do przesunięcia rozkładu każdego atrybutu w celu uzyskania średniej równej zero i odchylenia standardowego wynoszącego jeden.

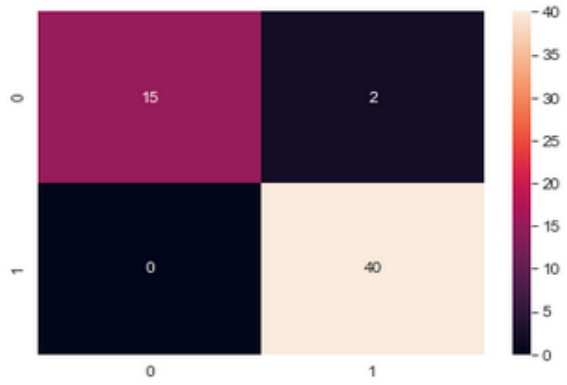
Standaryzacja danych w programie:

```
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaled_data_cancer = scaler.fit_transform(data_cancer['data'])
```

2.4 Analiza jakości uzyskanego modelu

Uzyskany przez nas model za pomocą regresji logistycznej daje następujące wyniki:

```
Model accuracy is 0.96
[[15  2]
 [ 0 40]]
```



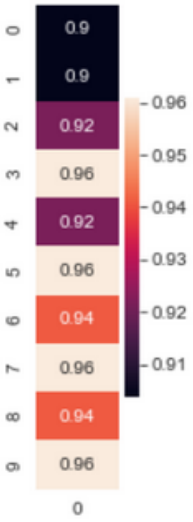
Wynik modelu jest przedstawiony za pomocą współczynnika dokładności. Obliczamy go dzieląc liczbę poprawnych dopasowań przez wszystkie próby dopasowania. W naszym przypadku dokładność wyniosła 96 %, co jest dosyć zadowalającym wynikiem. Macierz przedstawia ilość poprawnych i niepoprawnych dopasowań. W pierwszym wierszu znajdują się informacje o 15 poprawnych dopasowaniach diagnozy zmiany złośliwej (0), oraz o 2 pomyłkach, gdzie model przypisał zmianie złośliwej etykietę zmiany łagodnej. W drugim wierszu znajdują się informację o 40 poprawnych dopasowaniach diagnozy zmiany łagodnej (1), oraz o braku pomyłek. Podczas tworzenia modelu 10% danych było zbiorem testującym, natomiast 90% zbiorem trenującym.

```
#Rozdzielenie danych na zbiór trenujący i testujący (90% do 10%)
cancer_train_data, cancer_test_data, \
cancer_train_target, cancer_test_target = \
train_test_split(scaled_data_cancer, data_cancer.target, test_size=0.1)
```

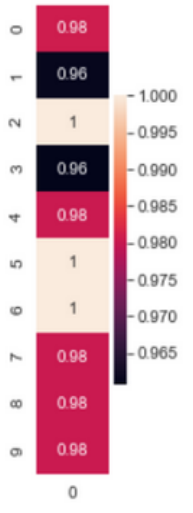
Dla naszych danych została również przeprowadzona sprawdzenie krzyżowe, czyli kroswalidacja, której wyniki są zbliżone do modelu wytrenowanego przez nas.

Wyniki przed skalowaniem danych i po kroswalidacji wyglądają następująco:

Przed:



Po:

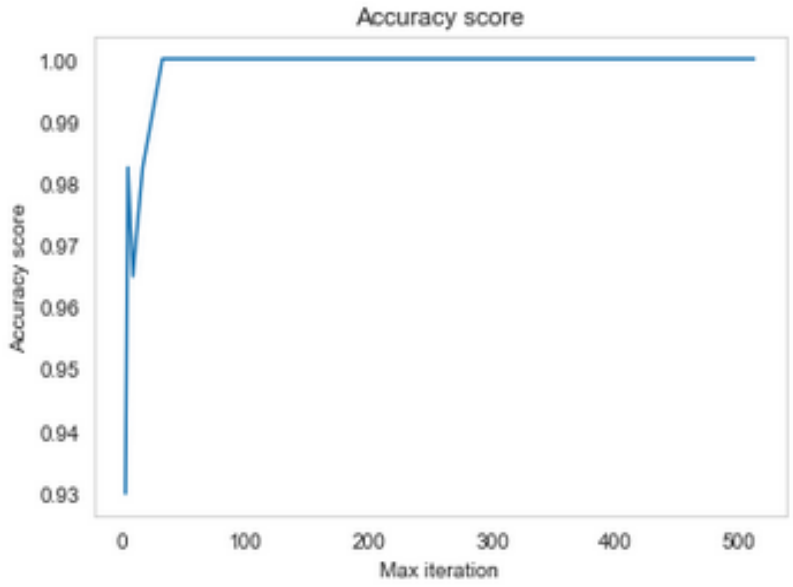


2.5 Badania symulacyjne

Badania wyniku zmienności naszego modelu rozpoczeliśmy od określania maksymalnej liczby iteracji dla algorytmu budującego klasyfikator.

Wykres przedstawiający dokładność modelu w zależności od maksymalnej liczby iteracji:

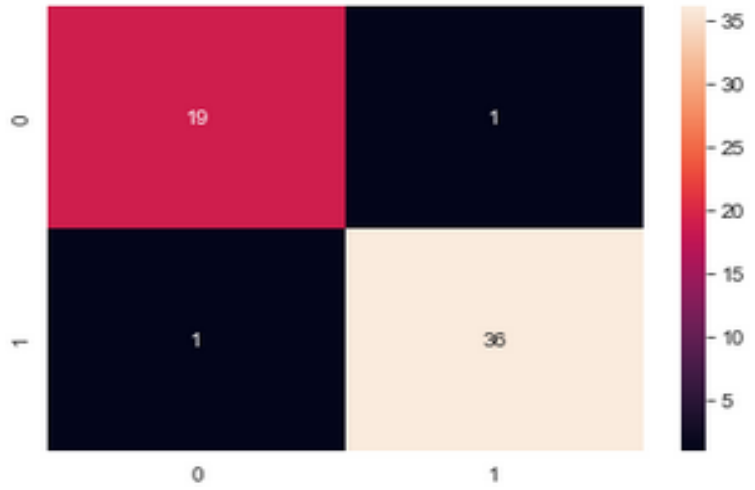
.



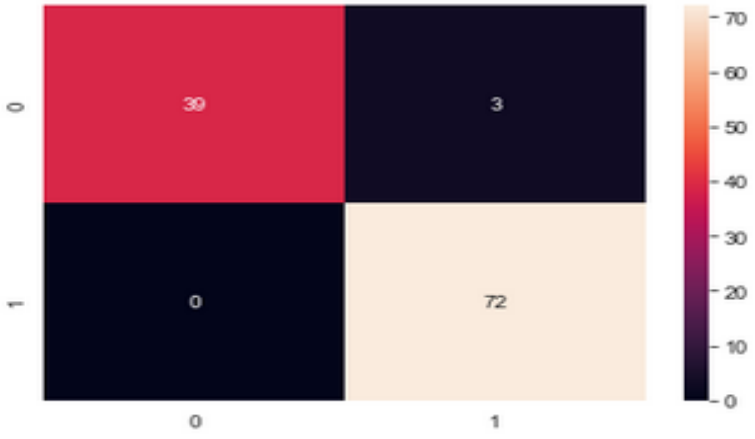
Maksymalne iteracje są potęgami liczby 2 do 512 włącznie.

Po przeanalizowaniu wykresu można wywnioskować, że dokładność wzrasta wraz ze wzrostem liczby iteracji, aż osiąga stu procentową skuteczność około 32 maksymalnych iteracji. Przeprowadzone zostały również badania wpływu rozkladu danych na zbiory testujące i trenujące:

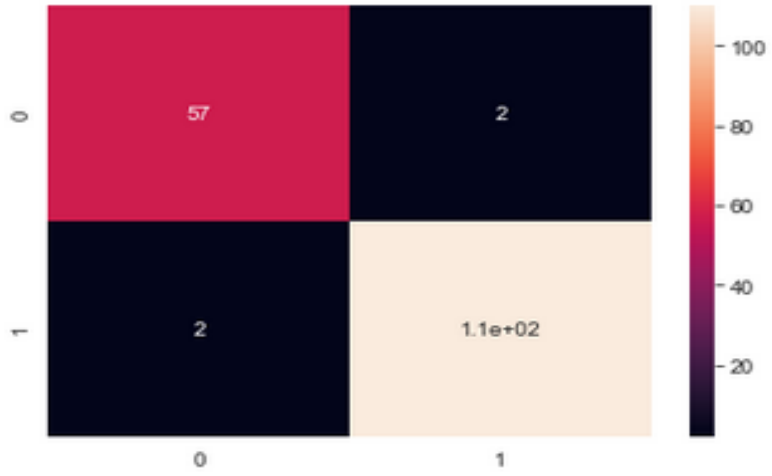
```
Model LogisticRegression basic - Test data = 10 %
Model accuracy is 0.96
[[19  1]
 [ 1 36]]
```



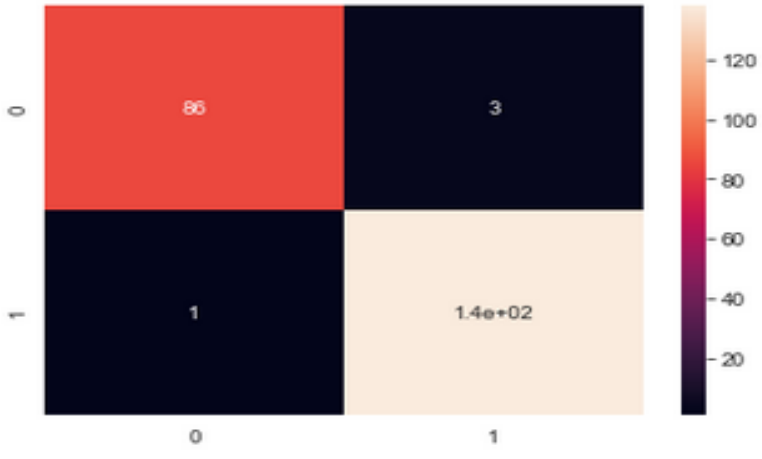

```
Model LogisticRegression basic - Test data = 20 %
Model accuracy is 0.97
[[39  3]
 [ 0 72]]
```



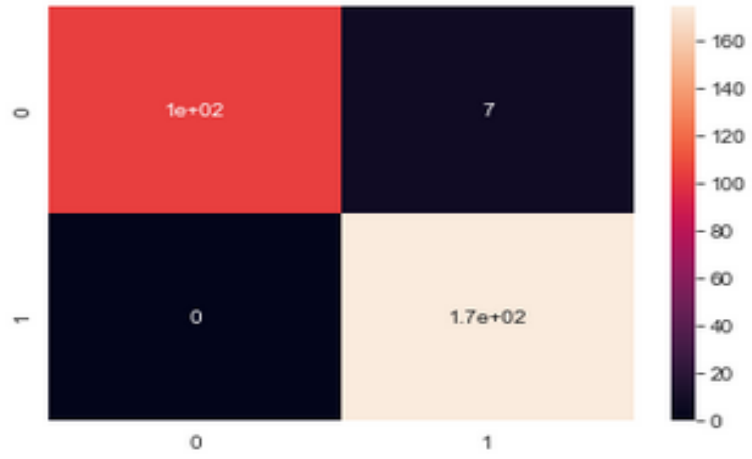
```
Model LogisticRegression basic - Test data = 30 %
Model accuracy is 0.98
[[ 57   2]
 [  2 110]]
```




```
Model LogisticRegression basic - Test data = 40 %
Model accuracy is 0.98
[[ 86   3]
 [  1 138]]
```

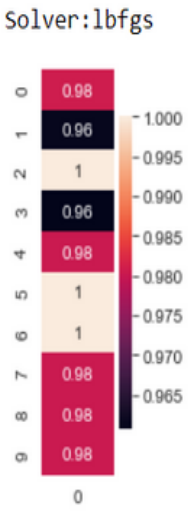


```
Model LogisticRegression basic - Test data = 50 %
Model accuracy is 0.98
[[104   7]
 [  0 174]]
```

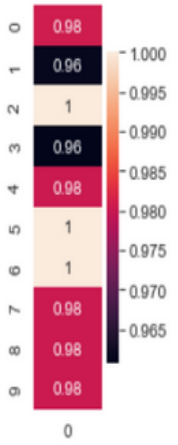


Zwiększając zbiór danych testujących wynik dla naszego modelu pozostaje bez znaczących zmian.

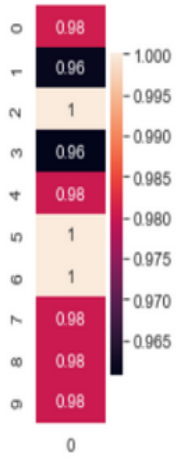
Sprawdzony został także wpływ rodzaju algorytmu optymalizacyjnego na rezultat działania modelu:

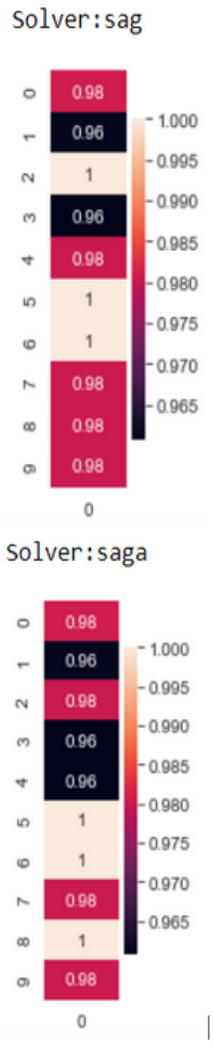


Solver:liblinear



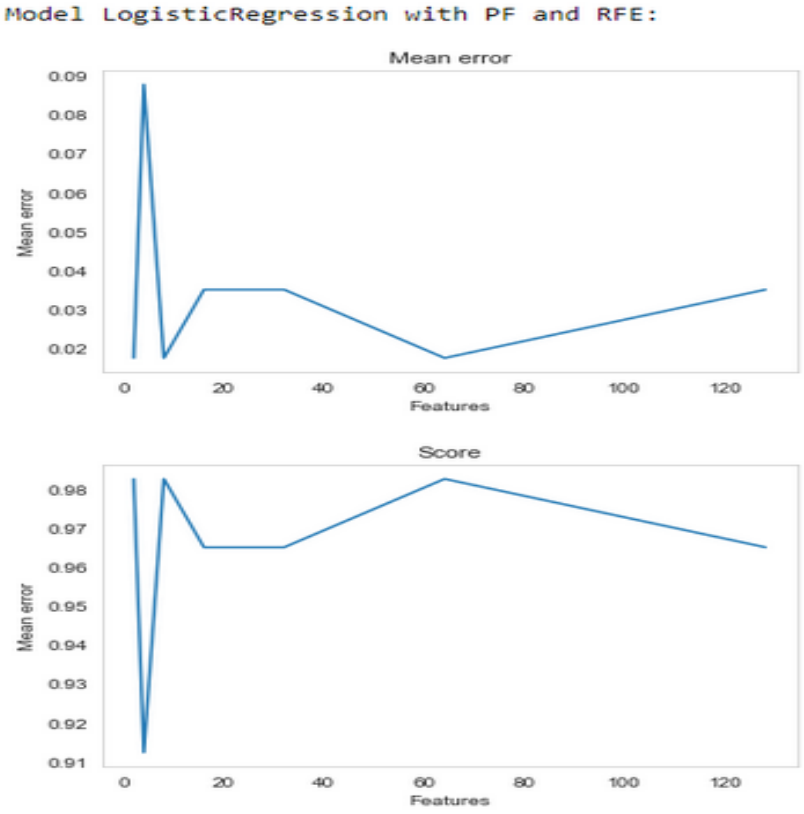
Solver:newton-cg





Zmiana algorytmu optymalizacyjnego nie wpłynęła znacząco na wynik modelu.

Zestaw danych został również poddany testom rekurencyjnej eliminacji cech. Jest to metoda opakowania, polegająca na szkoleniu estymatora na początkowym zestawie atrybutów, a następnie usuwaniu najmniej ważnych atrybutów. Procedura ta jest powtarzana rekurenycjnie, aż do momentu osiągnięcia potrzebnej nam ilości atrybutów do wyboru. Określone zostały również cechy wielomianowe, które zostały obliczone poprzez podniesienie istniejących cech do potęgi. Na wykresach przedstawiona jest zależnośc średniego błędu i wyniku od liczby wybranych atrybutów. W obu przypadkach można zauważyc spadek jakości w przypadku wybrania więcej niż 60 atrybutów.



Rozdział 3

Podsumowanie

Wytrenowany klasyfikator bardzo poprawnie radzi sobie z klasyfikowaniem analizowanych przez nas danych. Skalowanie zbioru danych znacząco polepszyło wynik modelu. Optymalną ilością maksymalnych iteracji algorytmu optymalizującego podczas budowania naszego modelu jest 100. Rodzaj algorytmu optymalizującego nie ma większego znaczenia na skuteczność klasyfikatora. Mimo zwiększania zbioru testującego i zmniejszania trenującego liniowa regresja radzi sobie bardzo dobrze z danymi kliniki.Rekurencyjna eliminacjia cech nie jest konieczna w trenowaniu naszego modelu. Należy pamiętać o tym, że nawet jeśli skuteczność modelu sięga stu procent, w dalszym ciągu diagnoza stawiana przez algorytm uczenia maszynowego powinna stanowić tylko jeden z kroków w ocenie pacjenta.

Dodatek A

Kod programu

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import cross_val_score
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt

#Cross walidacja dla naszych danych
def cross_valid_data(iteration):
    print("\nCross walidacja:")
    scores = cross_val_score(LogisticRegression(), cancer_train_data,
    cancer_train_target, cv=iteration)
    plt.figure(figsize=(1, 5))
    sns.heatmap(scores[:, np.newaxis],annot=True)
    plt.show()

#Przetestowanie naszego modelu dla określonych przez nas elementów
#- porównanie wyniku modelu z rzeczywistym wynikiem
def compare_model_resault_to_real_value(first_item, last_item):
    for i in range(first_item, last_item):

     prediction = logistic_regression.predict(cancer_test_data[i,:].reshape(1,-1))
     print("\nModel predicted for client {0} value {1}".format(i, prediction))
     print("Real value for client \"{0}\" is {1}".format(i, cancer_test_target[i]))
     #Wyświetlenie
     print(logistic_regression.predict_proba(cancer_test_data[i,:].reshape(1,-1)))
```

```
#Załadowanie danych
data_cancer = load_breast_cancer()

#Skalowanie daty
scaler = StandardScaler()
scaled_data_cancer = scaler.fit_transform(data_cancer['data'])

#Rozdzielenie danych na zbiór trenujący i testujący (90\% do 10\%)
cancer_train_data, cancer_test_data, \
cancer_train_target, cancer_test_target = \
train_test_split(scaled_data_cancer, data_cancer.target, test_size=0.1)

#Informacje o rozmiarze podzielonych danych
print("Training dataset:")
print("Train_data:", cancer_train_data.shape)
print("Train_target:", cancer_train_target.shape)

print("Testing dataset:")
print("Test_data:", cancer_test_data.shape)
print("Test_target:", cancer_test_target.shape)

#Trenowanie modelu z pomocą regresji logistycznej
logistic_regression = LogisticRegression()
logistic_regression.fit(cancer_train_data, cancer_train_target)

#Skuteczność naszego modelu
print("Model LogisticRegression basic:")
acc = accuracy_score(cancer_test_target, logistic_regression \
.predict(cancer_test_data))
print("Model accuracy is {0:0.2f}".format(acc))
conf_matrix = confusion_matrix(cancer_test_target, logistic_regression \
.predict(cancer_test_data))
print(conf_matrix)

sns.heatmap(conf_matrix,annot=True)
plt.show()

compare_model_resault_to_real_value(0,10)
cross_valid_data(10)

----------------------------------------

from sklearn.datasets import load_breast_cancer
import pandas as pd
```

```
import seaborn as sns
import matplotlib.pyplot as plt

data_cancer = load_breast_cancer()
pd_data = pd.DataFrame(data_cancer['data'], columns= data_cancer.feature_names)

fig = plt.figure()

for i,b in enumerate(list(bcdf.columns[0:30])):
    i +=1
    ax = fig.add_subplot(5,6,i)
    ax.hist(pd_data[b], label = 'Value', stacked = True, color= 'b')
    ax.set_title(b)

plt.tight_layout()
plt.show()

-----------------------------------------

\%matplotlib inline

import matplotlib.pyplot as plt
import seaborn as sb

sb.pairplot(pd_data, diag_kind="kde")

-----------------------------------------

# POLY FEATURES AND RFE
from sklearn.preprocessing import PolynomialFeatures
from sklearn.feature_selection import RFE
from sklearn.metrics import mean_squared_error

x = []
y_mean = []
y_score = []

def PolynomialFeatures_for_LogisticRegression(degree):
    print("\nModel LogisticRegression with PF and RFE:")

    pt = PolynomialFeatures(degree, )

    cancer_train_poly = pt.fit_transform(cancer_train_data)
    cancer_test_poly = pt.fit_transform(cancer_test_data)
```

```
    #logistic_regression_poly = LogisticRegression(solver='lbfgs', max_iter=300)
    #logistic_regression_poly.fit(credit_clients_train_poly, cancer_train_target)

    i=2

    while i <= 128:
        sel_ = RFE(estimator=LogisticRegression(solver='lbfgs', max_iter=300), \
        n_features_to_select=i)
        sel_.fit(cancer_train_poly, cancer_train_target)
        x.append(i);
        y_mean.append(mean_squared_error(cancer_test_target, \
        sel_.predict(cancer_test_poly)))
        y_score.append(sel_.score(cancer_test_poly, cancer_test_target))
        i*=2

   #acc = accuracy_score(cancer_test_target, sel_.predict(credit_clients_test_poly))
   #print("Model accuracy is {0:0.2f}".format(acc))

   #conf_matrix = confusion_matrix(cancer_test_target
   #, sel_.predict(credit_clients_test_poly))
   #print(conf_matrix)

PolynomialFeatures_for_LogisticRegression(2)

plt.plot(x,y_mean)
plt.title("Mean error")
plt.grid()
plt.xlabel("Features")
plt.ylabel("Mean error")
plt.show()

plt.plot(x,y_score)
plt.title("Score")
plt.grid()
plt.xlabel("Features")
plt.ylabel("Mean error")
plt.show()

-----------------------------------------

\%matplotlib inline
```

```
tab = ['newton-cg', 'lbfgs', 'liblinear', 'sag', 'saga']


for sol in tab:
    scores = cross_val_score(LogisticRegression(solver=sol), \
    cancer_train_data, cancer_train_target, cv=10)
    plt.figure(figsize=(1,4))
    sns.heatmap(scores[:, np.newaxis],annot=True)
    print("Solver:" + sol)
    plt.show()

----------------------------------------

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

pd_data = pd.DataFrame(data_cancer['data'], columns= data_cancer.feature_names)

sns.heatmap(pd_data.corr(),cmap="YlGnBu")
plt.show()

----------------------------------------

pd_data['Diagnosis'] = data_cancer.target
fig, ax = plt.subplots(figsize=(12,12))
sns.heatmap(pd_data.corr()[['Diagnosis']], cmap='YlGnBu', annot=True);
ax.invert_yaxis()

----------------------------------------

for size in range (1,6):
    cancer_train_data, cancer_test_data, \
    cancer_train_target, cancer_test_target = \
    train_test_split(scaled_data_cancer, data_cancer.target, test_size=(size/10))
    #Trenowanie modelu z pomocą regresji logistycznej
    logistic_regression = LogisticRegression()
    logistic_regression.fit(cancer_train_data, cancer_train_target)

    #Skuteczność naszego modelu
    print("Model LogisticRegression basic - Test data = {} \%".format(size*10))
    acc = accuracy_score(cancer_test_target, \
    logistic_regression.predict(cancer_test_data))
    print("Model accuracy is {0:0.2f}".format(acc))
```

```
    conf_matrix = confusion_matrix(cancer_test_target, \
    logistic_regression.predict(cancer_test_data))
    print(conf_matrix)

    sns.heatmap(conf_matrix,annot=True)
    plt.show()

----------------------------------------

i = 2
x = []
y_acc = []

while i <= 512:
    x.append(i)
    logistic_regression = LogisticRegression(max_iter=i)
    logistic_regression.fit(cancer_train_data, cancer_train_target)
    acc = accuracy_score(cancer_test_target, \
    logistic_regression.predict(cancer_test_data))
    y_acc.append(acc)
    i *= 2

----------------------------------------

plt.plot(x, y_acc)
plt.title("Accuracy score")
plt.grid()
plt.xlabel("Max iteration")
plt.ylabel("Accuracy score")

def PolynomialFeatures_for_LogisticRegression(degree):
    print("\nModel LogisticRegression with PF and RFE:")

    pt = PolynomialFeatures(degree, )

    cancer_train_poly = pt.fit_transform(cancer_train_data)
    cancer_test_poly = pt.fit_transform(cancer_test_data)


    logistic_regression_poly = LogisticRegression(solver='lbfgs', max_iter=50)
    logistic_regression_poly.fit(cancer_train_poly, cancer_train_target)

    x.append(i);
    y_mean.append(mean_squared_error
```

```
(cancer_test_target, logistic_regression_poly.predict(cancer_test_poly)))
    y_score.append
(logistic_regression_poly.score(cancer_test_poly, cancer_test_target))

for i in range(1,5):
    PolynomialFeatures_for_LogisticRegression(i)

plt.plot(x,y_mean)
plt.title("Mean error")
plt.grid()
plt.xlabel("Features")
plt.ylabel("Mean error")
plt.show()

plt.plot(x,y_score)
plt.title("Score")
plt.grid()
plt.xlabel("Features")
plt.ylabel("Mean error")
plt.show()
```