



---

# Sieci neuronowe

---

Raport ćw. 1-4

AUTOR

**Szymon Sawczuk**

nr albumu: **260287**

kierunek: **Informatyka Stosowana**

*19 października 2023*

# Spis treści

<b>1 Ćwiczenie 1 - Analiza danych</b>	<b>2</b>
1.1 Biblioteki użyte w tym ćwiczeniu . . . . .	2
1.2 Eksploracja danych . . . . .	2
1.3 Przygotowanie macierzy cech liczbowych . . . . .	6
1.4 Wnioski . . . . .	7
<b>2 Ćwiczenie 2 - Model sieci neuronowej wykorzystującej regresję logistyczną</b>	<b>8</b>
2.1 Implementacja modelu . . . . .	8
2.2 Test modelu na bazie danych z ćwiczenia 1 . . . . .	10
2.3 Podsumowanie wyników . . . . .	11
<b>Literatura</b>	<b>11</b>

# 1 Ćwiczenie 1 - Analiza danych

Celem ćwiczenia było zapoznanie (bądź przypomnienie) się z bibliotekami i narzędziami, które wykorzystywane są do uczenia maszynowego, eksploracji danych oraz ewaluacji sieci neuronowych, a także analiza zbioru danych wykorzystywanych do tego i dalszych ćwiczeń. (*Heart Disease Dataset*, 1988)

## 1.1 Biblioteki użyte w tym ćwiczeniu

W tym ćwiczeniu wykorzystałem 3 biblioteki dostępne dla języka python:

- pandas - biblioteka pozwalająca na łatwe tworzenie zbiorów danych oraz ich eksplorację i modyfikację
- matplotlib - do tworzenia wykresów danych
- seaborn - dla zaawansowanych wizualizacji danych np. mapy ciepła

Zapoznałem się także z bibliotekami, które będą potrzebne do kolejnych ćwiczeń: numpy (biblioteka do operacji na wielowymiarowych tabelach/macierzach), Scikit-learn (dająca implementacje algorytmów do preprocessing'u oraz algorytmów uczenia maszynowego).

## 1.2 Eksploracja danych

Po załadowaniu danych poprzez prosty skrypt podany na stronie zbioru danych (*Heart Disease Dataset*, 1988), uzyskałem 14 kolumn w zbiorze danych:

- age (liczbowa) - wiek osoby (lata)
- sex (kategoryczna) - płeć osoby (0 - kobieta, 1 - mężczyzna)
- cp (kategoryczna) - typ bólu klatki piersiowej (wartości 1-4)
- trestbps (liczbowa) - ciśnienie krwi w spoczynku (mmHg)
- chol (liczbowa) - poziom cholesterolu w surowicy (mg/dl)
- fbs (kategoryczna) - poziom cukru we krwi na czczo (0 - nie, 1 - tak)
- restecg (kategoryczna) - wynik elektrokardiografii w spoczynku (0 - normalny, 1 - ST-T anormalność, 2 - hipertrofia)
- thalach (liczbowa) - maksymalne tętno osiągnięte podczas testu wysiłkowego
- exang (kategoryczna) - dławica wysiłkowa (0 - nie, 1 - tak)
- oldpeak (liczbowa) - depresja odcinka ST wywołana przez ćwiczenia w stosunku do odpoczynku
- slope (kategoryczna) - nachylenie odcinka ST podczas ćwiczeń (0 - wnoszące, 1 - płaskie, 2 - opadające)
- ca (liczbowa) - liczba głównych naczyń (0-3), podczas badania fluoroskopowego
- thal (kategoryczna) - rodzaj defektu (3 - normalny, 6 - uleczony defekt, 7 - odwracalny defekt)
- num - obecność choroby serca (0 - brak, 1,2,3,4 - obecność (czym większa wartość tym poważniejsza choroba))

Dane składają się z 303 próbek oraz 13 cech, kolumna num określa nam obecność choroby serca, albo jej brak.

```
heart_data.sample(10)
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	num
261	58	0	2	136	319	1	2	152	0	0.0	1	2.0	3.0	3
169	45	0	2	112	160	0	0	138	0	0.0	2	0.0	3.0	0
40	65	0	4	150	225	0	2	114	0	1.0	2	3.0	7.0	4
241	41	0	2	126	306	0	0	163	0	0.0	1	0.0	3.0	0
12	56	1	3	130	256	1	2	142	1	0.6	2	1.0	6.0	2
259	57	1	2	124	261	0	0	141	0	0.3	1	0.0	7.0	1
201	64	0	4	180	325	0	0	154	1	0.0	1	0.0	3.0	0
246	58	1	4	100	234	0	0	156	0	0.1	1	1.0	7.0	2
99	48	1	4	122	222	0	2	186	0	0.0	1	0.0	3.0	0
255	42	0	3	120	209	0	0	173	0	0.0	2	0.0	3.0	0

Rysunek 1: 10 losowych przykładowych danych po wczytaniu

Pierwszą rzeczą jaką zbadałem było zbalansowanie danych względem liczby próbek w klasie.

```
heart_data["num"].value_counts()
```

num	count
0	164
1	55
2	36
3	35
4	13

Rysunek 2: Liczba próbek w klasach zbioru

Wyniki wskazują na brak zbalansowania danych pod względem liczby próbek na klasy. 164 próbki (około 54%) są próbkami zdrowych pacjentów (bez wykazanych problemów z sercem). Natomiast osób z zdiagnozowanymi najpoważniejszymi chorobami serca (klasa 4) jest tylko 4%. Rozwiązanie tego problemu wytłumaczone zostanie w następnym podrozdziale.

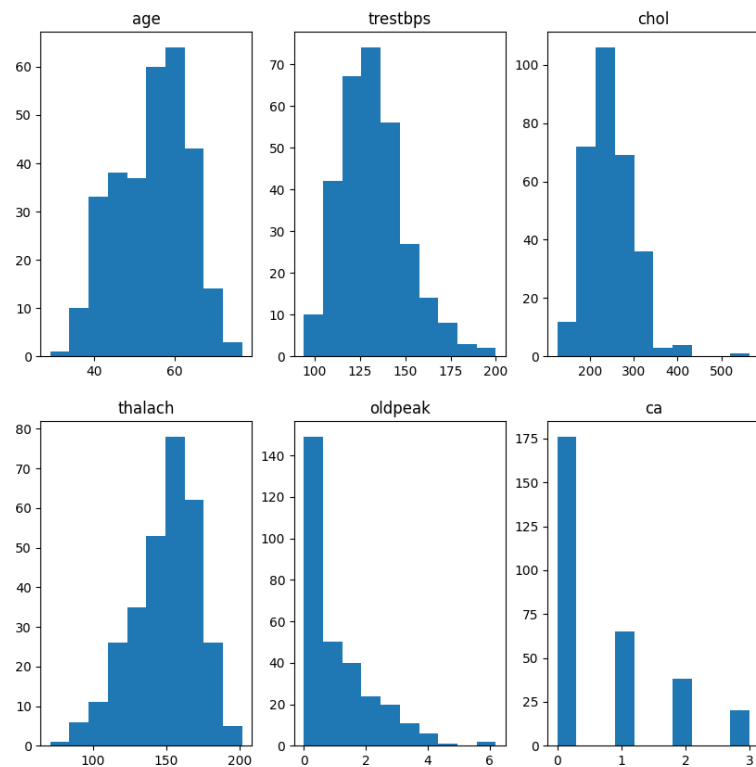
Następnym elementem badanym były wartości średnie oraz odchylenia standardowe cech liczbowych zbioru.

<pre>heart_data[num_features].mean()</pre>	<pre>heart_data[num_features].std()</pre>																												
<table> <thead> <tr> <th></th><th>mean</th></tr> </thead> <tbody> <tr> <td>age</td><td>54.438944</td></tr> <tr> <td>trestbps</td><td>131.689769</td></tr> <tr> <td>chol</td><td>246.693069</td></tr> <tr> <td>thalach</td><td>149.607261</td></tr> <tr> <td>oldpeak</td><td>1.039604</td></tr> <tr> <td>ca</td><td>0.672241</td></tr> </tbody> </table>		mean	age	54.438944	trestbps	131.689769	chol	246.693069	thalach	149.607261	oldpeak	1.039604	ca	0.672241	<table> <thead> <tr> <th></th><th>std</th></tr> </thead> <tbody> <tr> <td>age</td><td>9.038662</td></tr> <tr> <td>trestbps</td><td>17.599748</td></tr> <tr> <td>chol</td><td>51.776918</td></tr> <tr> <td>thalach</td><td>22.875003</td></tr> <tr> <td>oldpeak</td><td>1.161075</td></tr> <tr> <td>ca</td><td>0.937438</td></tr> </tbody> </table>		std	age	9.038662	trestbps	17.599748	chol	51.776918	thalach	22.875003	oldpeak	1.161075	ca	0.937438
	mean																												
age	54.438944																												
trestbps	131.689769																												
chol	246.693069																												
thalach	149.607261																												
oldpeak	1.039604																												
ca	0.672241																												
	std																												
age	9.038662																												
trestbps	17.599748																												
chol	51.776918																												
thalach	22.875003																												
oldpeak	1.161075																												
ca	0.937438																												

Rysunek 3: Wartości średnie oraz odchylenia standardowe cech liczbowych

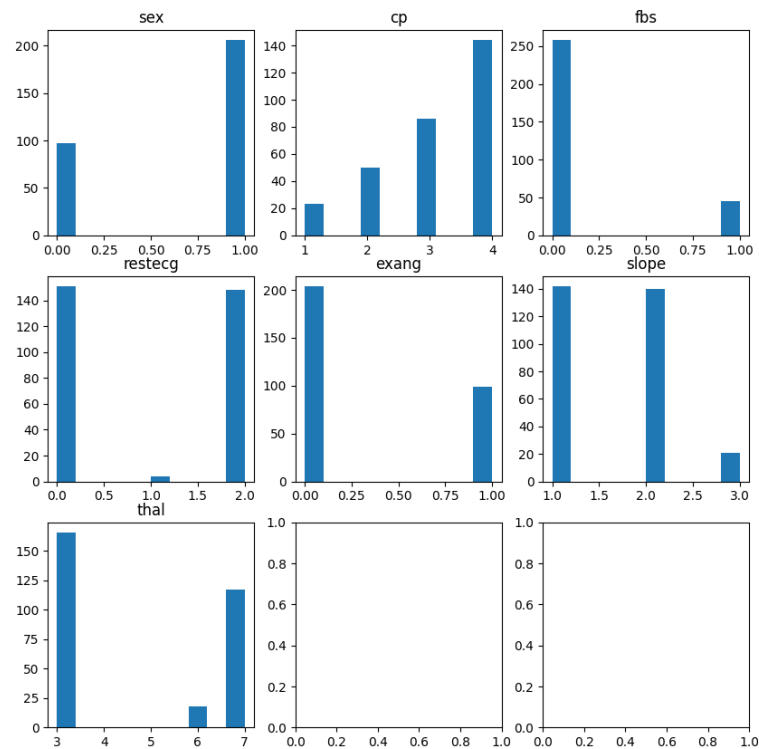
Dla przykładu średnia wartość wieku w zbiorze danych wynosi 54 lata, a około 70% danych mieści się między wiekiem 45, a 63. Widać już tutaj potencjalne rozkłady niektórych cech np. wieku. Natomiast histogramy wykazane poniżej wykazują, że cechy wieku, ciśnienia krwi w spoczynku, poziomu cholesterolu oraz maksymalnego osiągniętego tętna układają się w przybliżeniu zgodnie z wykresem

Gausa, zatem posiadają one rozkłady normalne. Dane depresji odcinka ST (oldpeak) oraz liczba naczyń zaobserwowanych poprzez fluoroskopię (ca) nie wykazują rozkładu normalnego, bardziej rozkład wykładniczy.



Rysunek 4: Histogramy cech liczbowych

Weźmy teraz pod lupę cechy katagoryczne i czy są one w przybliżeniu równomierne.



Rysunek 5: Histogramy cech katagorycznych

Na powyższych histogramach cech kategorycznych nie widać, aby jakkolwiek cecha miała zrównoważone dane. Najbliżej jednak takiego rozkładu równomiernego są cechy danych elektrokardiograficznych (restecg) oraz nachylenie odcinka ST (slope). Z danych nierównomiernych widać np. że większą ilością badanych byli mężczyźni.

W zbiorze odnalazłem 2 cechy, które posiadają wartości puste jest to ca oraz thal. Łącznie wartości pustych jest 6.

```
heart_data[heart_data["ca"].isnull()]
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	num
166	52	1	3	138	223	0	0	169	0	0.0	1	NaN	3.0	0
192	43	1	4	132	247	1	2	143	1	0.1	2	NaN	7.0	1
287	58	1	2	125	220	0	0	144	0	0.4	2	NaN	7.0	0
302	38	1	3	138	175	0	0	173	0	0.0	1	NaN	3.0	0

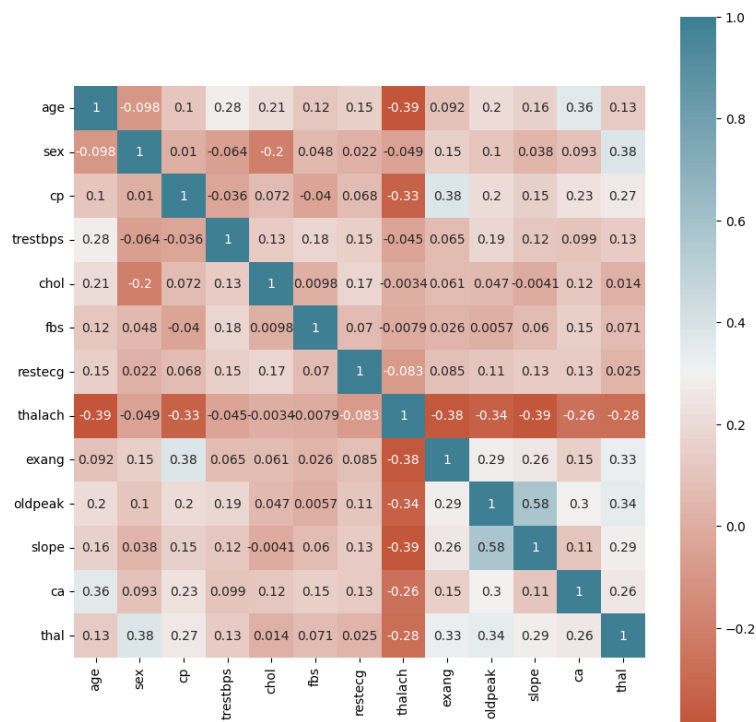
  

```
heart_data[heart_data["thal"].isnull()]
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	num
87	53	0	3	128	216	0	2	115	0	0.0	1	0.0	NaN	0
266	52	1	4	128	204	1	0	156	1	1.0	2	0.0	NaN	2

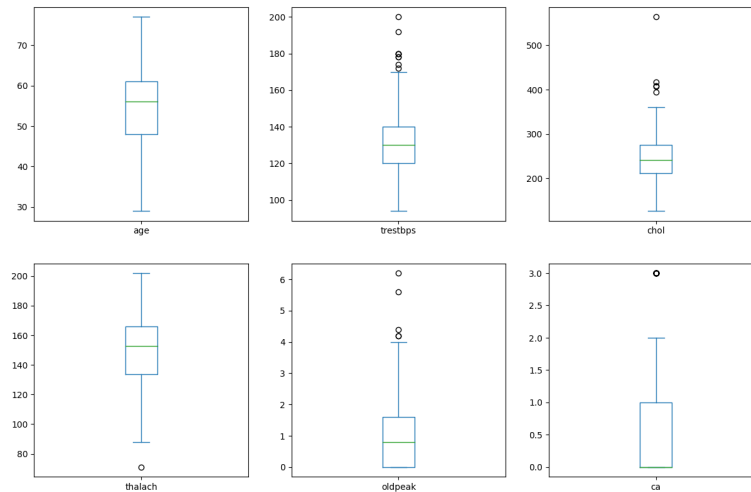
Rysunek 6: Wartości puste zbioru

Jest wiele sposobów na rozwiązanie tego problemu np. uzupełnienie brakujących danych w sposób sztuczny używając mediany, albo algorytmu k-najbliższych sąsiadów (*K najbliższych sąsiadów*, 2022). Natomiast z powodu małej ilości danych brakujących (około 2% danych), możemy najprościej usunąć te dane ze zbioru, bez znaczącej utraty informacji.



Rysunek 7: Wykres ciepła dla korelacji między cechami

Mapy ciepła zamieszczone powyżej pokazują poziom korelacji cech danych. Przy wysokich wartościach korelacji możnaby rozważyć usunięcie jednej z tych cech np. (slope i oldpeak), mogłoby to pomóc w uzyskaniu lepszych wyników nauczania. Warto jednak rozważyć także sens merytoryczny tych dwóch cech, czy jednak nie są one znaczące dla całego modelu.



Rysunek 8: Wykresy pudełkowe dla cech liczbowych

Powyższe wykresy pudełkowe wskazują nam na rozłożenie wartości danych cech. Widzimy, że dane ca, oldpeak są w mniejszym zakresie niż np. wiek. Takie dane o małych zakresach mogą zostać przykryte w niektórych modelach przez cechy o większych zakresach. Warto też przyrzeć się danym odbiegającym od kwartyli cechy (można rozważyć ich usunięcie).

### 1.3 Przygotowanie macierzy cech liczbowych

Po wyciągnięciu cech liczbowych ze zbioru danych zająłem się rozwiązaniem problemu braku zbilansowania próbek względem klasyfikacji zbioru. Zdecydowałem na naprawę braku zbalansowania próbek poprzez zmniejszenie klasyfikacji do klasyfikacji binarnej (0 - zdrowy, 1 - choroba serca), ponieważ klasy 1-4 oznaczały inne stopnie problemów z sercem, które można na potrzeby modelu budowanego zmniejszyć do tej samej klasyfikacji. Owe rozwiązanie pozwoliło także na zmniejszenie ilości danych potrzebnych do usunięcia, aby klasyfikacje były zbilansowane. W wyniku uzyskałem zmniejszony zbiór do 274 próbek, ale ze zbilansowanymi próbkami względem klas.

```
# repairing of the imbalance in classification and removing null values
df["num"] = df["num"].replace([2, 3, 4], 1) #change classes to binary classification
print(df["num"].value_counts())

# get null values of ca and remove them
null_idx = df[df["ca"].isnull()].index
print(null_idx)
df = df.drop(null_idx)
df = df.reset_index(drop=True)
print(df["num"].value_counts())

# get null values of thal and remove them
null_idx = df[df["thal"].isnull()].index
print(null_idx)
df = df.drop(null_idx)
df = df.reset_index(drop=True)
print(df["num"].value_counts())

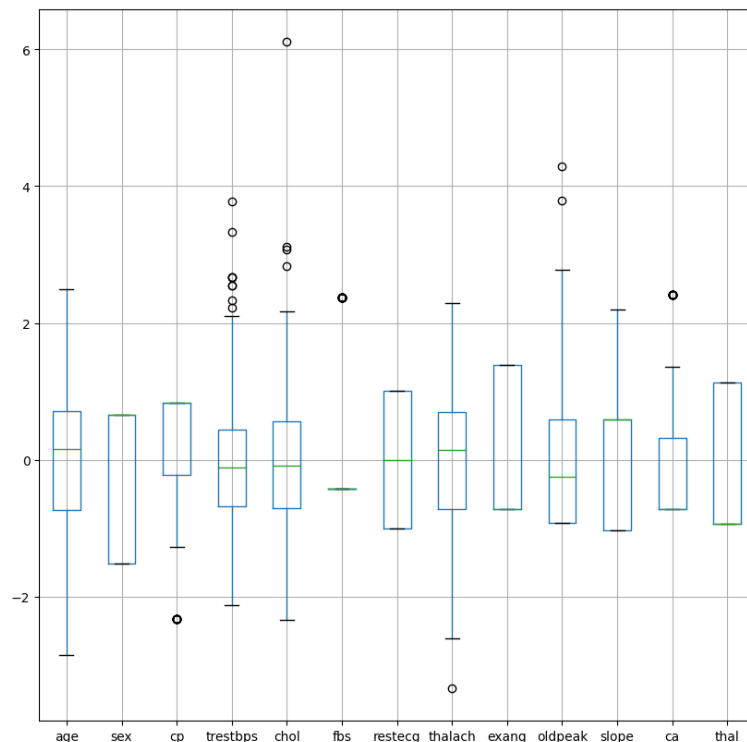
# balance classes to same amount 138
random_idx = df.query("num == 0").sample(df["num"].value_counts()[0] - df["num"].value_counts()[1]).index
df = df.drop(random_idx)
df = df.reset_index(drop=True)
print(df["num"].value_counts())
```

Rysunek 9: Kod naprawiający zbilansowanie próbek

Usunięcie tych danych pozwoliło także na pozbycie się wartości pustych dla cechy ca oraz thal.

Następnie rozwiązałem problem różnych zakresów cech liczbowych poprzez standaryzację cech, w taki sposób zachowane pozostały rozkłady owych cech. Standaryzację wykonałem za pomocą wzoru  $z = \frac{x - \mu}{\sigma}$ , gdzie:

- $x$  – zmienna niestandardyzowana,
- $\mu$  – średnia z populacji,
- $\sigma$  – odchylenie standardowe populacji.



Rysunek 10: Wykresy pudełkowe cech po standaryzacji

Wynikiem wszystkich tych operacji jest gotowa macierz cech liczbowych z przykładami, którą można wykorzystać do dalszych ćwiczeń.

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	heart_disease
0	0.934497	0.658004	-2.320666	0.718175	-0.275798	2.378535	1.00564	0.056964	-0.715498	1.015065	2.195678	-0.724267	0.613001	0
1	1.380193	0.658004	0.832371	1.551988	0.746442	-0.418716	1.00564	-1.748946	1.392528	0.343664	0.587080	2.409142	-0.927937	1
2	1.380193	0.658004	0.832371	-0.671514	-0.352948	-0.418716	1.00564	-0.845991	1.392528	1.266841	0.587080	1.364672	1.126647	1
3	-1.962526	0.658004	-0.218641	-0.115638	0.052090	-0.418716	-1.00564	1.647885	-0.715498	2.022167	2.195678	-0.724267	-0.927937	0
4	0.154530	0.658004	-1.269653	-0.671514	-0.217935	-0.418716	-1.00564	1.260904	-0.715498	-0.243812	-1.021519	-0.724267	-0.927937	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
269	0.265954	-1.514202	0.832371	0.440237	-0.121497	-0.418716	-1.00564	-1.103978	1.392528	-0.747363	0.587080	-0.724267	1.126647	1
270	-1.071134	0.658004	-2.320666	-1.227389	0.322116	-0.418716	-1.00564	-0.718997	-0.715498	0.091889	0.587080	-0.724267	1.126647	1
271	1.491617	0.658004	0.832371	0.662587	-1.047299	2.378535	-1.00564	-0.330016	-0.715498	1.938242	0.587080	1.364672	1.126647	1
272	0.265954	0.658004	0.832371	-0.115638	-2.243126	-0.418716	-1.00564	-1.447981	1.392528	0.091889	0.587080	0.320202	1.126647	1
273	0.265954	-1.514202	-1.269653	-0.115638	-0.217935	-0.418716	1.00564	1.088913	-0.715498	-0.915213	0.587080	0.320202	-0.927937	1

Rysunek 11: Wynikowa macierz cech liczbowych

## 1.4 Wnioski

- Analiza danych pozwala nam na lepsze zrozumienie zbioru, a także naprawę problemów w zbiorze, które mogą spowodować gorsze wyniki naszego modelu.
- Warto zwrócić uwagę na zbalansowanie klasyfikacji w próbkach, gdyż brak owego zbalansowania może nauczyć model rozpoznawania poprawnej klasyfikacji tylko w kilku z nich (tych klas, których jest najwięcej).
- Innymi wartymi uwagi problemami jakie mogą pojawić się w zbiorze są brakujące dane, nierówne zakresy cech, bądź zbyt duża korelacja danych.
- Warto zwrócić uwagę na rozkłady cech w zbiorze, ponieważ może nam to pomóc w wyborze odpowiedniego modelu do rozwiązania naszego zadania.
- Analiza danych to proces iteracyjny, który nie raz wymaga wielu kroków, warto wspomóc się bibliotekami np. dla języka python, które pomagają nam na np. szybszą operację na danych oraz różne wizualizacje zbioru danych.

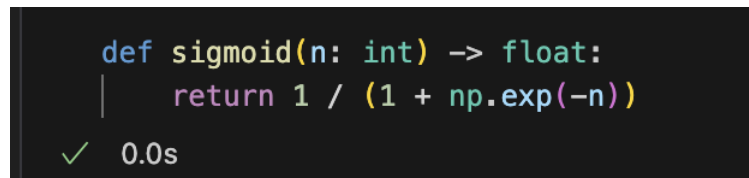


## 2 Ćwiczenie 2 - Model sieci neuronowej wykorzystującej regresję logistyczną

Celem ćwiczenia było przygotowanie własnoręcznego modelu sieci neuronowej wykorzystującej regresję logistyczną, a później przetestowanie działania wykorzystując dane przygotowane w ćwiczeniu 1. Do zadania wykorzystałem biblioteki numpy (do operacji na macierzach), pandas (do przygotowania danych jak w ćwiczeniu 1), matplotlib (wykonanie wykresów przedstawiający wyniki uczenia się modelu) oraz scikit learn (do łatwej weryfikacji wyników sklearn.metrics)

### 2.1 Implementacja modelu

Wyjście modelu opiera się o funkcję sigmoidu:



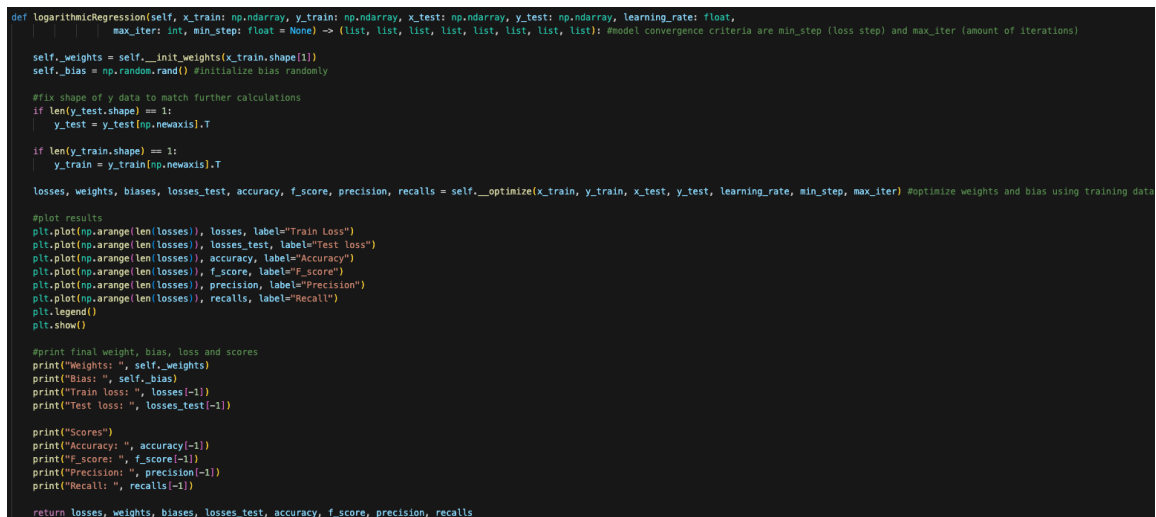
```
def sigmoid(n: int) -> float:
    return 1 / (1 + np.exp(-n))
```

✓ 0.0s

Rysunek 12: Funkcja sigmoidu

Warto jednak zauważyć, że z powodu limitu typu float przy większym wejściu (np. 42) funkcja ta zwróci nam wartość 1, co może powodować problemy przy obliczaniu dalszym logarytmu naturalnego z  $(1 - \text{sigmoid})$ , ponieważ  $\ln(0)$  jest nieokreślony. Przy danych z poprzedniego ćwiczenia, nie pojawi się ten problem z powodu wcześniejszej standaryzacji danych (dane mają małe wartości).

Zaimplementowałem klasę `LogarithmicRegression`, która posiada atrybuty prywatne wag oraz bias'u. Główna metoda wywołująca uczenie modelu nazywa się `logarithmicRegression` i przyjmuje ona dane treningowe oraz testowe, współczynnik uczenia, oraz warunki zbieżności modelu (pewna maksymalna liczba iteracji i wystarczająco mała zmiana funkcji kosztu w danej iteracji i)



```
def logarithmicRegression(self, x_train: np.ndarray, y_train: np.ndarray, x_test: np.ndarray, y_test: np.ndarray, learning_rate: float,
                           max_iter: int, min_step: float = None) -> (list, list, list, list, list, list, list, list): #model convergence criteria are min_step (loss step) and max_iter (amount of iterations)

    self._weights = self._init_weights(x_train.shape[1])
    self._bias = np.random.rand() #initialize bias randomly

    #fix shape of y data to match further calculations
    if len(y_test.shape) == 1:
        y_test = y_test[np.newaxis].T

    if len(y_train.shape) == 1:
        y_train = y_train[np.newaxis].T

    losses, weights, biases, losses_test, accuracy, f_score, precision, recalls = self._optimize(x_train, y_train, x_test, y_test, learning_rate, min_step, max_iter) #optimize weights and bias using training data

    #plot results
    plt.plot(np.arange(len(losses)), losses, label="Train Loss")
    plt.plot(np.arange(len(losses)), losses_test, label="Test Loss")
    plt.plot(np.arange(len(losses)), accuracy, label="Accuracy")
    plt.plot(np.arange(len(losses)), f_score, label="F_score")
    plt.plot(np.arange(len(losses)), precision, label="Precision")
    plt.plot(np.arange(len(losses)), recalls, label="Recall")
    plt.legend()
    plt.show()

    #print final weight, bias, loss and scores
    print("Weights: ", self._weights)
    print("Bias: ", self._bias)
    print("Train loss: ", losses[-1])
    print("Test loss: ", losses_test[-1])

    print("Scores")
    print("Accuracy: ", accuracy[-1])
    print("F_score: ", f_score[-1])
    print("Precision: ", precision[-1])
    print("Recall: ", recalls[-1])

    return losses, weights, biases, losses_test, accuracy, f_score, precision, recalls
```

Rysunek 13: Główna metoda klasy `LogarithmicRegression`

W pierwszym kroku inicjalizuję wartości wag oraz bias'u poprzez wartości losowe. Następnie po przygotowanie wymiarów macierzy wartości y (wyniki przykładów) dokonuję optymalizacji wag i bias'u:

```
def __optimize(self, x_train: np.ndarray, y_train: np.ndarray, x_test: np.ndarray, y_test: np.ndarray, learning_rate: float,
               min_step: float, max_iter: int) -> (list, list, list, list, list, list, list, list):
    losses = []
    weights = []
    biases = []
    losses_test = []

    accuracy = []
    f_score = []
    precision = []
    recalls = []
    for index in range(max_iter): #learn for max_iter

        old_train_loss = self.__cross_entropy_loss(self.weights, x_train, y_train, self.bias)

        #calculate new weight and bias using step
        self.weights = self.weights - learning_rate * self.__gradient_weights(self.weights, x_train, y_train, self.bias)
        self.bias = self.bias - learning_rate * self.__gradient_bias(self.weights, x_train, y_train, self.bias)

        #calculate loss of training and testing data
        new_train_loss = self.__cross_entropy_loss(self.weights, x_train, y_train, self.bias)
        new_test_loss = self.__cross_entropy_loss(self.weights, x_test, y_test, self.bias)

        #append to helper lists
        losses.append(new_train_loss)
        weights.append(self.weights)
        biases.append(self.bias)
        losses_test.append(new_test_loss)

        #calculate scores for each iteration
        y_pred = self.predict(x_test)
        accuracy.append(metrics.accuracy_score(y_test, y_pred))
        f_score.append(metrics.f1_score(y_test, y_pred))
        precision.append(metrics.precision_score(y_test, y_pred))
        recalls.append(metrics.recall_score(y_test, y_pred))

        if min_step is not None and index > 0 and abs(old_train_loss - new_train_loss) <= min_step: #if change of loss is smaller or equal than min_step when stop learning
            break

    return losses, weights, biases, losses_test, accuracy, f_score, precision, recalls
```

Rysunek 14: Metoda optymalizacji

Wagi zostają przesunięte o wartość gradientu wyliczanego dla całego zbioru danych zgodnie z wzorem  $w' = w - learning\_rate * gradient\_wag$ :

```
def __gradient_weights(self, weights: np.ndarray, x: np.ndarray, y: np.ndarray, bias: float) -> np.ndarray:
    return -np.dot(x.T, (y - sigmoid(np.dot(x, weights) + bias))) / x.shape[0] #calculate gradient of weights using whole training data
```

Rysunek 15: Gradient wag

Dla bias'u natomiast dane x wynoszą 1 więc możemy uprościć wyliczenie gradientu do:

```
def __gradient_bias(self, weights: np.ndarray, x: np.ndarray, y: np.ndarray, bias: float) -> np.ndarray:
    return np.sum(-(y - sigmoid(np.dot(x, weights) + bias))) / x.shape[0] #calculate gradient of bias using whole training data
```

Rysunek 16: Gradient bias'u

$x.shape[0]$  oznacza ilość przykładów w macierzy danych treningowych.

Następnie obliczony jest koszt po obliczeniu nowych wag i bias'u:

```
def __cross_entropy_loss(self, weights: np.ndarray, x: np.ndarray, y: np.ndarray, bias: float) -> np.ndarray:
    return np.sum(-y*np.log(sigmoid(np.dot(x, weights) + bias)) - (1 - y)*np.log(1 - sigmoid(np.dot(x, weights) + bias))) / x.shape[0] #calculate loss using whole training data
```

Rysunek 17: Wzór kosztu funkcji

Dla każdej iteracji obliczony zostaje także wynik, względem predykcji wyników dla danych testowych:

Sigmoid daje nam prawdopodobieństwo w przedziale (0, 1), więc wyniki należy zaokrąglić do pełnego 0, albo 1 (próg 0.5).

```
def predict(self, x_test: np.ndarray) -> np.ndarray:

    y_pred = sigmoid(np.dot(x_test, self._weights) + self._bias) #calculate sigmoid for input data

    #change possibility from sigmoid to 1 or 0
    y_pred[y_pred >= 0.5] = 1
    y_pred[y_pred < 0.5] = 0

    return y_pred
```

Rysunek 18: Metoda predykcji danych

Na koniec sprawdzany jest warunek zbieżności modelu - wystarczająco mała zmiana funkcji kosztu w danej iteracji i.

Wyniki optymalizacji oraz przebieg kosztów w trakcie iteracji są przedstawiane za pomocą wykresu oraz zwracane jako wynik metody `logarithmicRegression`.

## 2.2 Test modelu na bazie danych z ćwiczenia 1

Po wczytaniu danych zgodnie z przebiegiem ćwiczenia 1, należy jeszcze podzielić dane na dane uczące i testowe. Wykonałem to w sposób losowy, dla pewnej ilości danych uczących względem testowych (np. 70% danych treningowych, 30% testowych):

```
def train_test_split(features, targets, percentage):
    choices = np.random.choice(range(features.shape[0]), size=(int(features.shape[0] * percentage/100),), replace=False)
    split = np.zeros(features.shape[0], dtype=bool)
    split[choices] = True

    return features[split], targets[split], features[~split], targets[~split]

features = result.loc[:, result.columns != "heart_disease"].to_numpy()
targets = result["heart_disease"].to_numpy()

x_train, y_train, x_test, y_test = train_test_split(features, targets, 70)
```

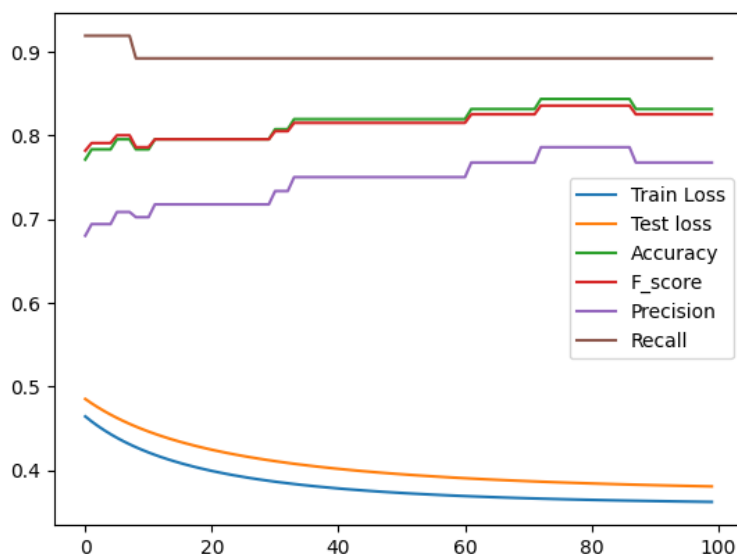
Rysunek 19: Funkcja dzieląca zbiór na dwa zbiory względem pewnego stosunku

Dla takich podzielonych danych oraz wartości współczynnika uczenia - 0.2, ilości iteracji - 200 oraz wartości progu małego spadku funkcji kosztu - 0.0001 uzyskujemy wyniki:

```
Weights: [[-0.11786781]
 [ 0.65462996]
 [ 0.43297219]
 [ 0.25495492]
 [ 0.04797949]
 [-0.10275015]
 [ 0.18658021]
 [-0.28018111]
 [ 0.51133107]
 [ 0.40773473]
 [ 0.40899077]
 [ 1.09021618]
 [ 0.66263916]]
Bias: 0.38287959027390495
Train loss: 0.3619743755584501
Test loss: 0.3804651798847365
Scores
Accuracy: 0.8313253012048193
F_score: 0.825
Precision: 0.7674418604651163
Recall: 0.8918918918918919
```

Rysunek 20: Wyniki testu modelu

Przebieg iteracji przedstawiony jest na poniższym wykresie:



Rysunek 21: Przebieg nauczania modelu

## 2.3 Podsumowanie wyników

Jak widać na powyższym wykresie funkcja kosztu przy nauczaniu modelu jest w każdej iteracji coraz mniejsza (minimalizacja funkcji kosztu). A miary takie, jak dokładność, precyzja i średnia harmoniczna zwiększają się wraz z przebiegiem uczenia (model coraz lepiej przewiduje nieznane nowe dane).

Dokładność, która sprawdza stosunek poprawnie przewidywanych wyników do wszystkich prób testujących, może nie być najlepszym wskaźnikiem jakości dla tego zadania (rozpoznawanie chorób serca), ponieważ te wyniki nieodgadnięte mogą należeć do osób chorych, albo model źle wskazał na chore osoby, które są tak naprawdę zdrowe. Lepiej zwrócić uwagę na precyzję oraz recall, które zwracają uwagę w wyliczeniu na klasy odgadnięte poprawnie oraz błędnie. Miara f1 natomiast jest średnią harmoniczną recall'a oraz precyzji. (*Precision, recall i F1 – miary oceny klasyfikatora*, 2019)

Model utworzony w tym zadaniu osiąga wyniki ok. 76% poprawnych przewidzianych wyników pozytywnych (precision), natomiast ilość elementów poprawnie rozpoznanych przez model wynosi 89% (recall). Średnia harmoniczna (f\_score) wskazuje, że uśredniony wynik dla naszego modelu to 82% poprawnie rozpoznanych przykładów.

## Literatura

*Heart Disease Dataset*. (1988, czerwiec). Retrieved from <https://archive.ics.uci.edu/dataset/45/heart+disease>

*K najbliższych sąsiadów*. (2022, czerwiec). Retrieved from [https://pl.wikipedia.org/wiki/K\\_najbli%C5%BCszych\\_s%C4%85siad%C3%B3w](https://pl.wikipedia.org/wiki/K_najbli%C5%BCszych_s%C4%85siad%C3%B3w)

*Precision, recall i F1 – miary oceny klasyfikatora*. (2019, listopad). Retrieved from <https://ksopyla.com/data-science/precision-recall-f1-miary-oceny-klasyfikatora/>