Wykrywanie chorób serca z wykorzystaniem algorytmów uczenia maszynowego

Projekt zaliczeniowy - Systemy Sztucznej Inteligencji

Wojciech Grzywocz grupa 3/5, Piotr Olasik grupa 3/5, Witold Pacholik grupa 3/5

27.05.2025

Część I

Opis programu

Projekt przedstawia wykorzystanie klasycznych algorytmów uczenia maszynowego w problemie klasyfikacji medycznej. Klasyfikacja ta polega na wykrywaniu czy dana osoba cierpi na chorobę serca czy też nie, na podstawie danych medycznych tej osoby. W projekcie wykorzystane zostały klasyczne algorytmy i techniki uczenia maszynowego takie jak na przykład Algorytm K-Najbliższych Sąsiadów (KNN), algorytm Naiwnego klasyfikatora Bayesa czy również algorytm PCA redukcji wymiarowości. Celem projektu było przetestowanie kilku klasyfikatorów na różnych wartościach parametrów celem znalezienia modelu, który najlepiej spełnia zadanie klasyfikacji pacjentów na podstawie ich danych medycznych.

Instrukcja obsługi

Projekt był uruchamiany i testowany w środowisku PyCharm z użyciem Pythona 3.8. Projekt składa się z plików: Data_exploration.py, evaluate.py, heart.csv, KNN.py, load_data.py, Naive_Bayes.py, model_neuron.py, preprocess.py, heart_failure_nn_model.h5, test_data.pkl, train.py. Projekt był przez nas uruchamiany w środowisku PyCharm w taki sposób iż kliknięte zostało Run wywołujące interpreter Pythona na każdym pliku .py z osobna, natomiast, co ważne, jako ostatni wywoływany był plik evaluate.py gdyż to on jest odpowiedzialny za finalną analizę danych i testy.

Dodatkowe informacje

Wymagania: Numpy, Pandas, Matplotlib, Seaborn, scikit-learn, tensorflow, keras, joblib
Opis plików:

- Data_exploration.py - zawiera klasę z metodami do analizy eksploracyjnej danych z bazy. Dodatkowo zawiera metodę podsumowującą, która pozwala na podsumowanie danych z bazy, poprzez wywołanie wszystkich pojedynczych metod tej klasy.
- evaluate.py - Najważniejszy plik pod kątem testowania. Wywołana zostaje tam analiza danych. Również w pliku tym wywołane i testowane są modele uczenia maszynowego i dokonane jest podsumowanie otrzymanych efektów dla tych modeli.
- heart.csv - Plik csv z danymi
- KNN.py - Plik z własnoręcznie napisanym algorytmem KNN. Metryka została dobrana na bazie wcześniejszych testów z gridSearch i knn z sklearn w pliku evaluate.py
- load_data.py - Zawiera metody do wczytania danych
- Naive_Bayes.py - Plik z własnoręcznie napisanym algorytmem Naiwnego Bayesa. Funkcja zastosowana w nim to funkcja dla rozkładu normalnego prawdopodobieństwa.

- model_neuron.py - Plik, który zawiera prosty model sieci neuronowej, która jest używana na koniec pliku evaluate jako porównanie skuteczności w kontekście wcześniejszych algorytmów.
- preprocess.py - Zawiera metody, które skalują dane i zwracają zbiór podzielony na X_train, X_test, Y_train i Y_test
- heart_failure_nn_model.h5 - zawiera zapisany prosty model sieci neuronowej, która jest używana na koniec pliku evaluate jako porównanie skuteczności w kontekście wcześniejszych algorytmów.
- test_data.pkl - zawiera zapisane dane testowe - X_test i Y_test
- train.py - plik, w którym wytrenowana zostaje sieć neuronowa, używana w celach porównawczych.

Część II

Opis i cel działania

Program służy do analizy porównawczej skuteczności klasycznych metod uczenia maszynowego wykorzystanych w problemie klasyfikacji medycznej. Klasyfikacja polega na predykcji chorób serca na podstawie danych medycznych około 1000 pacjentów.
Problem występowania chorób układu krążenia jest o tyle poważny iż na dzień dzisiejszy choroby te stanowią główną przyczynę zgonów na świecie. Jak doskonale wiemy najlepszym sposobem leczenia jest odpowiednia profilaktyka i przeciwdziałanie wystąpienia problemów kardiologicznych. Pomimo, że można podać wiele czynników zdrowotnych i związanych ze stylem życia, które wpływają na wystąpienie problemów, to jednak występowanie chorób układu krążenia niesie za sobą pewną losowość i nie przewidywalność. Dlatego tak ważne jest korzystanie z nowoczesnych technologii, jak sztuczna inteligencja, które pozwalają na wcześniejsze przewidzenie problemów zdrowotnych na podstawie gromadzonych danych zdrowotnych pacjentów, bo jak wiemy lepiej jest zapobiegać niż leczyć. Z perspektywy informatyki natomiast kluczowe jest opracowanie jak najskuteczniejszych modeli sztucznej inteligencji, które nie tylko będą osiągały wysokie wartości accuraccy, lecz także recall na klasie chorych, gdyż zależy nam na jak najmniejszej liczbie przeoczonych przypadków. Natomiast błędne sklasyfikowanie pacjenta zdrowego jako chorego nie będzie rażącym błędem gdyż wyselekcjonuję grupę podejrzanych o zachorowanie, których po dokładniejszym przebadaniu i wykluczeniu choroby będzie można monitorować jako grupę ryzyka podejrzaną o potencjalne zachorowanie w przyszłości.

Użyte algorytmy

Użyte zostały algorytmy KNN, Naive Bayes, PCA, gridSearch, i prosta sieć neuronowa w celach sprawdzenia metryk.
Kod własnoręcznie napisanego KNN, który wykorzystuje metrykę opracowaną w oparciu o wcześniejsze testy na gridSearch:

```
1  # Hand-made KNN:
2  import math
3  import pandas as pd
4
5  class KNNClass:
6      def metric(self, a, b):
7          sum = 0
8          if len(a) == len(b):
9
10             # Canberra metric
11             for i in range(len(a)):
12                 denom = math.fabs(a[i]) + math.fabs(b[i])
13                 if denom != 0:
14                     sum += math.fabs(a[i] - b[i]) / denom
15                 else:
16                     sum += 1
17
18             return sum
```

```
        else:
            print("Different dimensions")

    def knn(self, X_train, Y_train, X_test, k):
        Y_prediction = []
        Y_train_np = Y_train.to_numpy()

        for X_test_vector in X_test:
            distances = []

            for i in range(len(X_train)):
                dist = self.metric(X_test_vector, X_train[i])
                distances.append((dist, Y_train_np[i]))

            distances.sort(key=lambda x: x[0])
            k_neighbors = distances[:k]  # list of tuples

            labels = [neighbor[1] for neighbor in k_neighbors]

            control = set(labels)
            most_common = ["", 0]
            for item in control:
                if (labels.count(item) > most_common[1]):
                    most_common = [item, labels.count(item)]

            Y_prediction.append(most_common[0])

        return pd.Series(Y_prediction)
```

Kod własnoręcznie napisanego Naive Bayes, który wykorzystuje funkcje gaussa dla rozkładu normalnego:

```
# Hand-made Naive Bayes
from math import sqrt, pi, exp
import numpy as np
import pandas as pd

class NaiveBayes:

    def gaussian(self, x, mean, std):
        if std == 0:
            return 1.0 if x == mean else 0.0
        exponent = exp(-((x - mean) ** 2.0) / (2.0 * std ** 2.0))
        return (1 / (sqrt(2.0 * pi) * std)) * exponent

    def summarize_by_class(self, X, y):
        summaries = {}
        for cls in np.unique(y):
            X_cls = X[y == cls]
            means = X_cls.mean(axis=0)
            stds = X_cls.std(axis=0)
            summaries[cls] = list(zip(means, stds))
        return summaries

```

```
23      def calculate_class_probabilities(self, summaries, input_vector,
            class_priors):
24          probabilities = {}
25          for cls, feature_summaries in summaries.items():
26              prob = class_priors[cls]
27              for i in range(len(input_vector)):
28                  mean, std = feature_summaries[i]
29                  prob *= self.gaussian(input_vector[i], mean, std)  # P(
                        Xi|Class)
30              probabilities[cls] = prob
31          return probabilities
32
33      def predict(self, summaries, input_vector, class_priors):
34          probs = self.calculate_class_probabilities(summaries,
                input_vector, class_priors)  # dictionary cls: prob
35          return max(probs, key=probs.get)
36
37      def get_predictions(self, X_train, Y_train, X_test):
38          summaries = self.summarize_by_class(X_train, Y_train)
39          class_priors = {cls: np.mean(Y_train == cls) for cls in np.
                unique(Y_train)}
40          preds = [self.predict(summaries, row, class_priors) for row in
                X_test]
41          return pd.Series(preds)
```

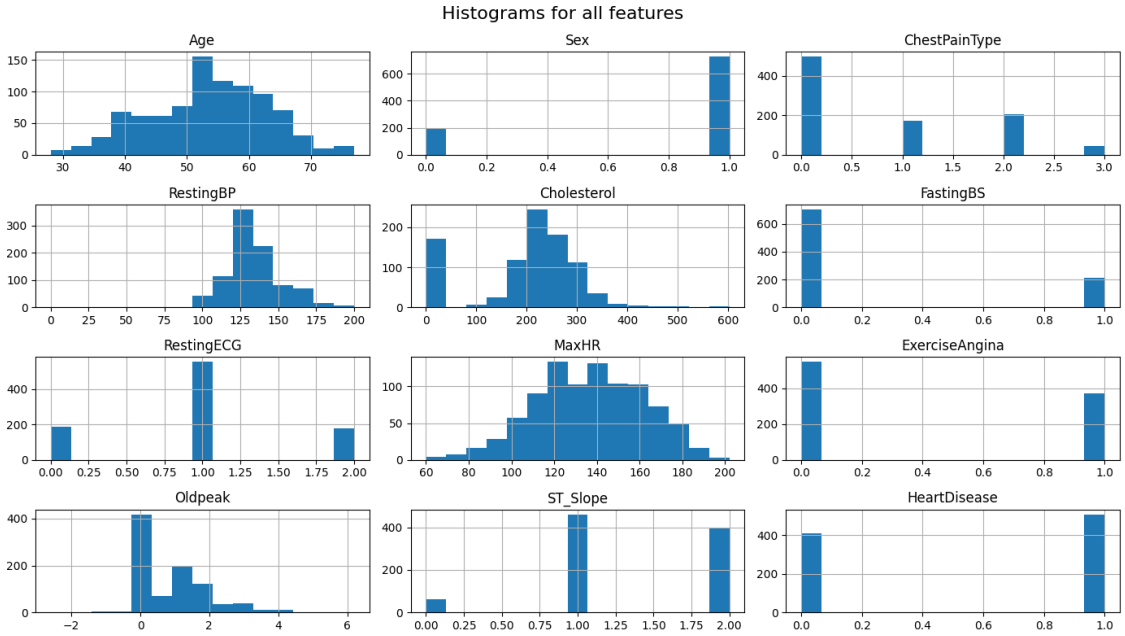
Implementacja

Projekt składa się z plików: Data_exploration.py, evaluate.py, heart.csv, KNN.py, load_data.py, Naive_Bayes.py, model_neuron.py, preprocess.py, heart_failure_nn_model.h5, test_data.pkl, train.py.
W pliku Data_exploration znajduje się klasa Data_exploration z metodami: info - do wyświetlania informacji o bazie danych, mising_values - do szukania niepełnych danych, head - która wyświetla pierwsze 5 wierszy bazy danych, description - która podaje opis parametrów statystycznych danych, unique_values - która wypisuje unikalne wartości z każdej kolumny, num_of_samples_per_class - która podaje ile mamy osób chorych, a ile zdrowych w bazie, histograms - wyświetla histogramy dla cech, boxplots - która wyświetla boxploty dla danych, distributions - która wyświetla rozkłady cech w zależności od klasy, pair_plots - która wyświetla pairploty dla danych, correlation_matrix - która wyświetla macierz korelacji dla danych, mean_values_per_class - wyświetla wartości średnie parametrów w zależności od klasy oraz kluczowa metoda summarize - która wywołuje wcześniejsze metody na zbiorze danych co pozwala na dogłębną analizę posiadanych danych.
Plik KKN.py zawiera własnoręcznie napisany algorytm KNN ze zmodyfikowaną metryką Canberra dobraną na bazie testów z użyciem gridSearch na KNN z sklearn.
Load_data.py zawiera 2 metody: jedną do wczytania danych do DataFrame'a, a drugą do przekształcenia kolumn typu object na wartości numeryczne z użyciem LabelEncodera.
Plik model_neuron zawiera funkcję, która tworzy i kompiluje prostą sieć neuronową.
Naive_Bayes.py zawiera kod własnoręcznie napisanego klasyfikatora Bayesa z funkcją Gaussa dla rozkładu normalnego.
preprocess.py zawiera funkcję, która przekształca dane z dataFrame poprzez zamianę kolumn

typu object na wartości numeryczne, a następnie dzieli zbiór na dane testowe i treningowe. Train.py z kolei trenuje sieć neuronową i zapisuje model oraz dane testowe do plików (heart_failure_nn_model.h5 oraz test_data.pkl).
W pliku evaluate.py wykonane zostaje główne działanie programu. Najpierw dokonana jest eksploracja i analiza danych, później podział na zbiory treningowe i testowe, następnie utworzone zostają dodatkowe zbiory z wykorzystaniem algorytmu PCA z 4, 6 i 8 składowymi. Następnie z użyciem gridSearch szukamy jak najlepszych parametrów dla KNN na każdym zbiorze danych (skalowane, PCA4, PCA6 i PCA8), później implementujemy nasz KNN i sprawdzamy jego działanie na różnych zbiorach danych w zależności od liczby sąsiadów. W następnym kroku testujemy algorytm Naiwnego Klasyfikatora Bayesa z sklearn na każdym zbiorze oraz nasz algorytm Klasyfikatora Bayesa na każdym zbiorze. Finalnie wykonujemy testy, jakie wyniki osiągnie prosta sieć neuronowa w porównaniu z naszymi modelami.

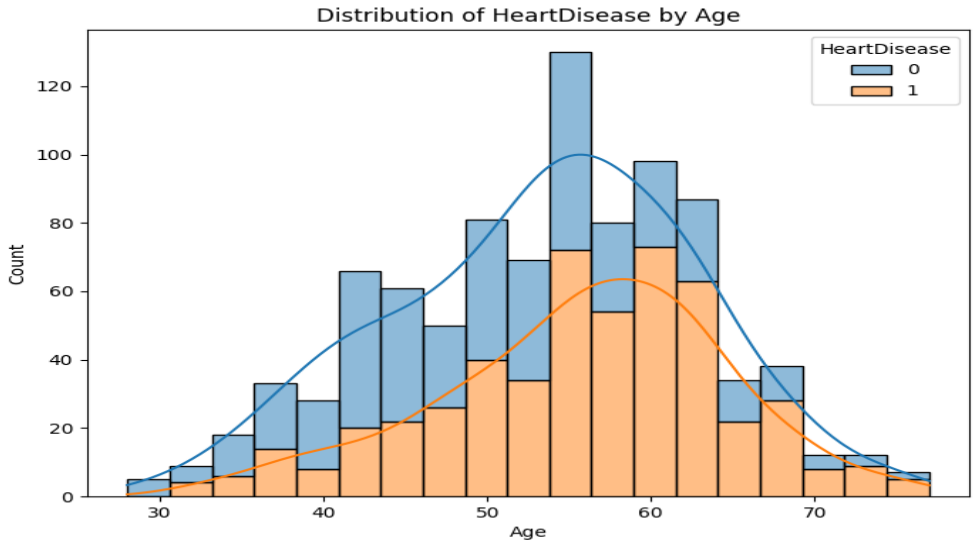
Testy i eksperymenty

Niżej umieszczone zostały najbardziej interesujące wnioski i wyniki uzyskane dzięki projektowi.
Na początek możemy zobaczyć rozkład danych:



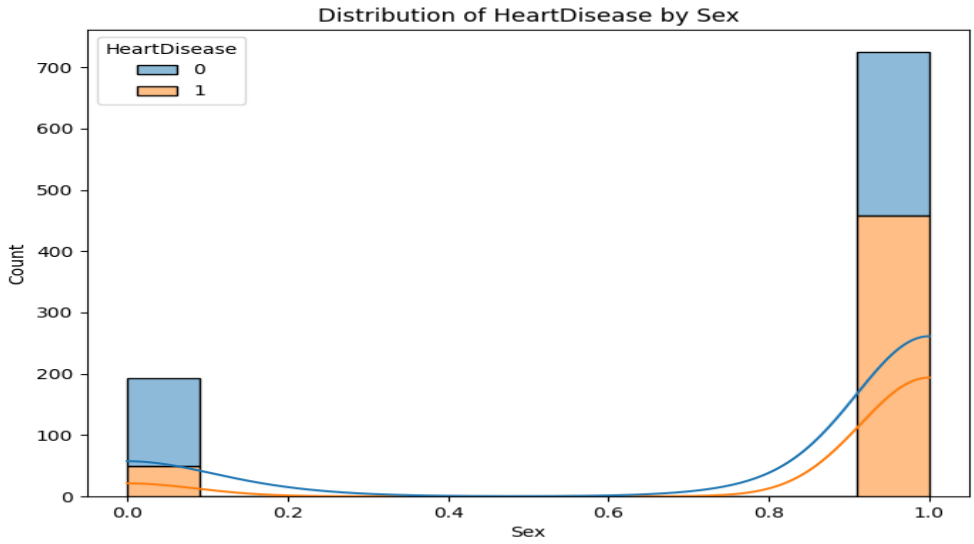
Rysunek 1: Histogram cech

Na rozkładach możemy zauważyć, że odsetek osób chorych wzrasta po 55 roku życia:



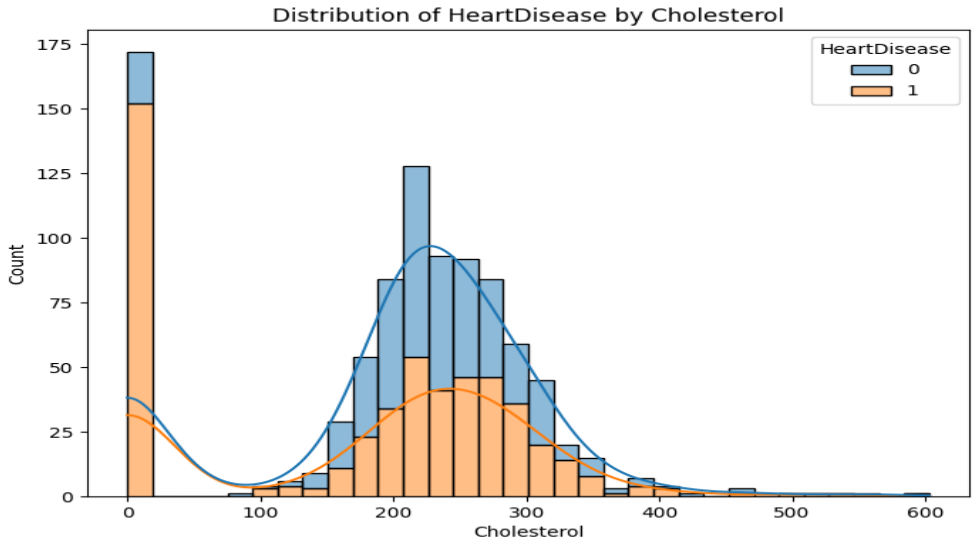
Rysunek 2: Rozkład klas względem wieku

Większy odsetek chorych wśród mężczyzn, jednak też więcej danych od mężczyzn niż od kobiet:



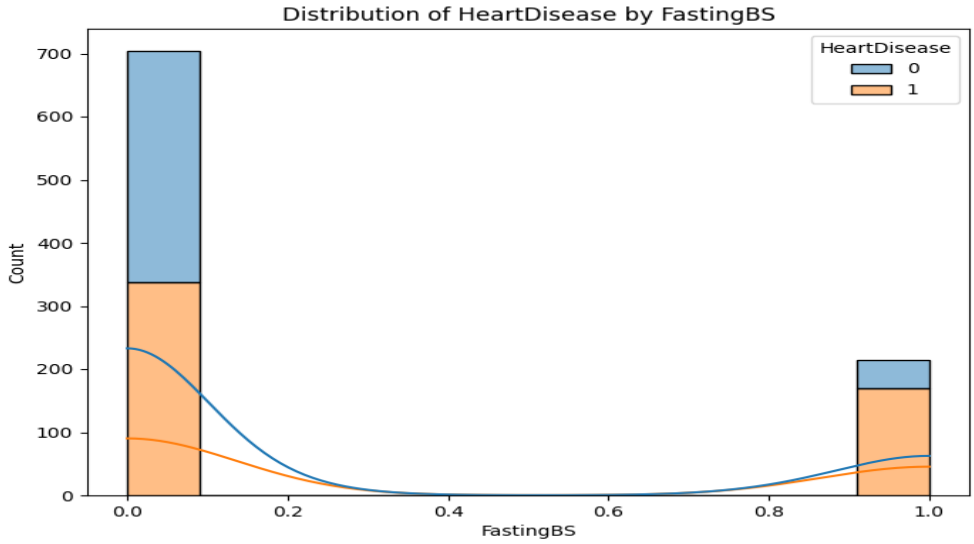
Rysunek 3: Rozkład klas względem płci

Im więcej cholesterolu, tym procent chorych wyższy:



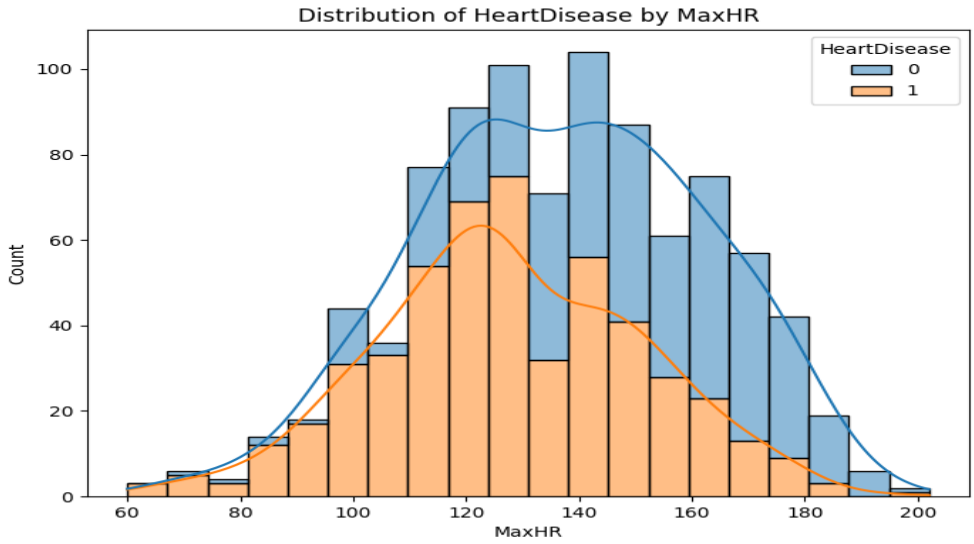
Rysunek 4: Rozkład klas względem poziomu cholesterolu

Ludzie z wysokim poziomem cukru, częściej chorują.



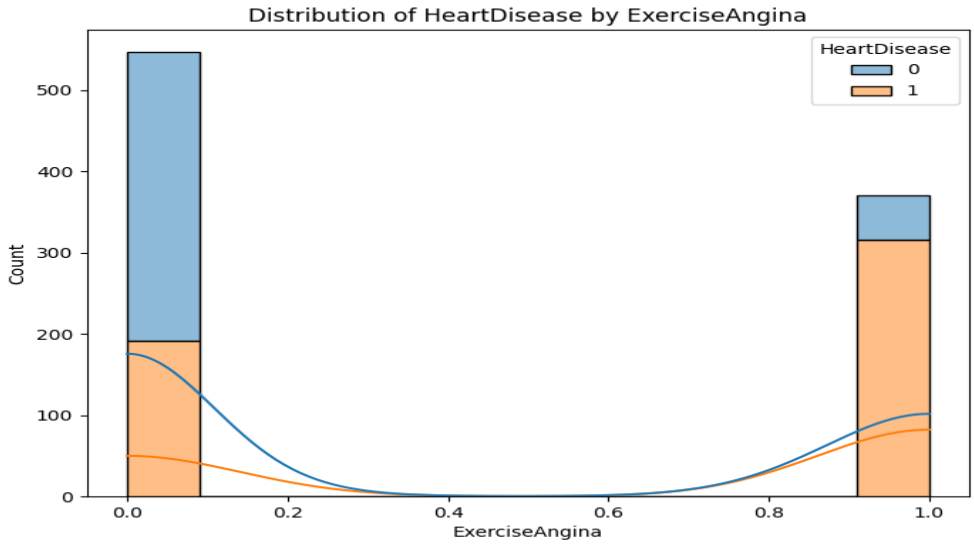
Rysunek 5: Rozkład klas względem poziomu cukru

Ludzie którzy osiągnęli niższe tętno maksymalne częściej chorowali:



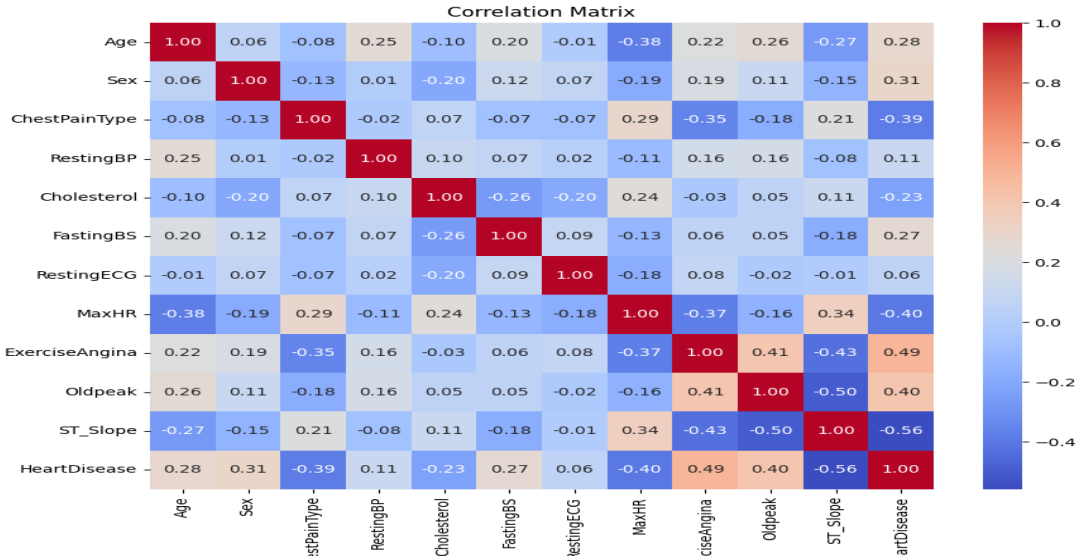
Rysunek 6: Rozkład klas względem maksymalnego tętna

Ludzie, którzy mieli dławicę piersiową spowodowaną wysiłkiem fizycznym częściej chorowali:



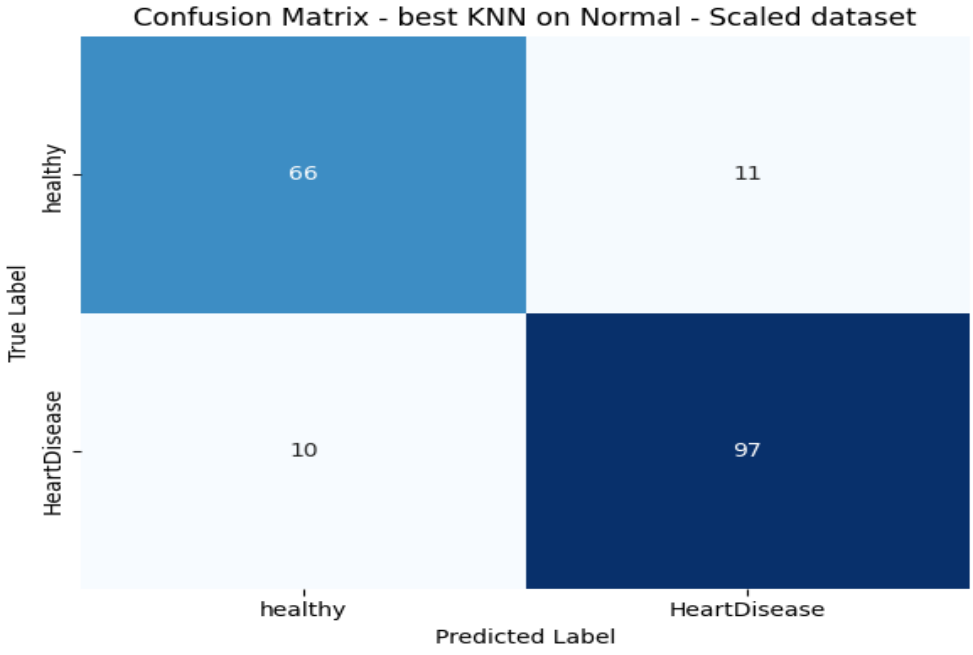
Rysunek 7: Rozkład klas a dławica piersiowa spowodowana wysiłkiem

Macierz korelacji:

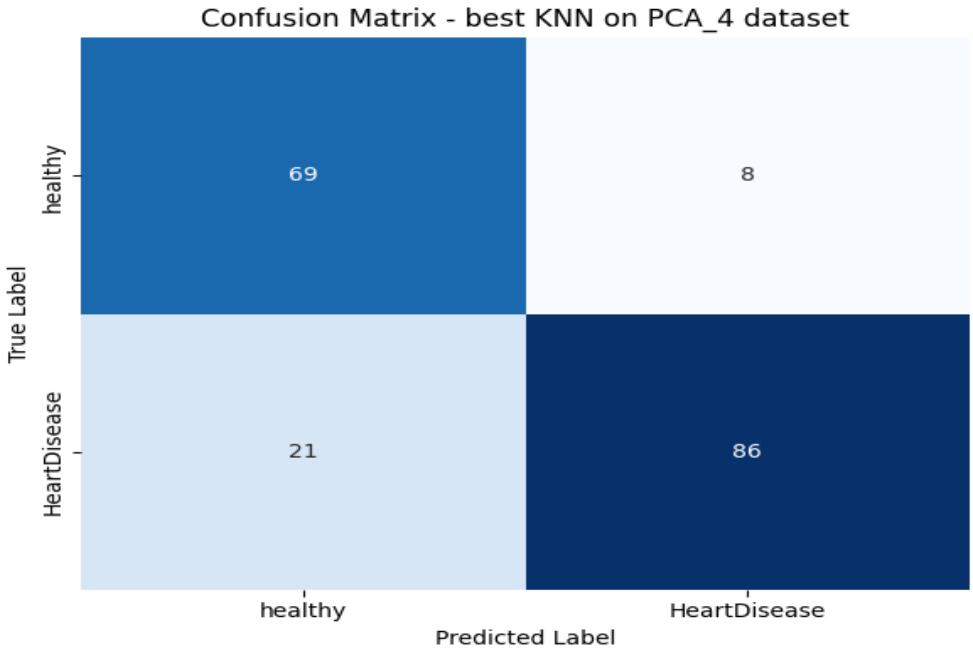


Rysunek 8: Macierz korelacji

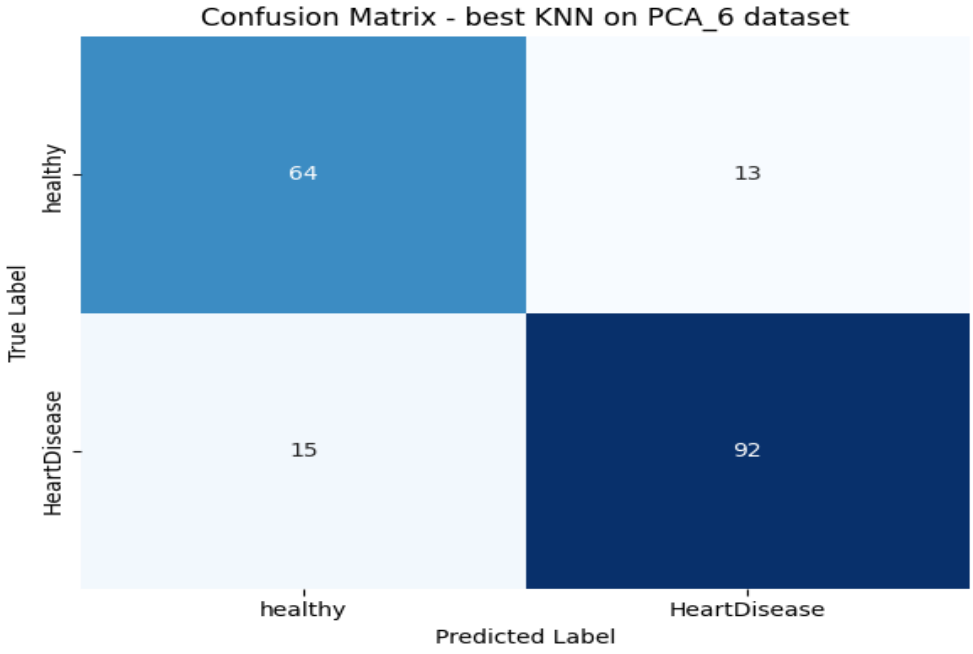
Wyniki KNN za pomocą Grid search na różnych zbiorach danych:



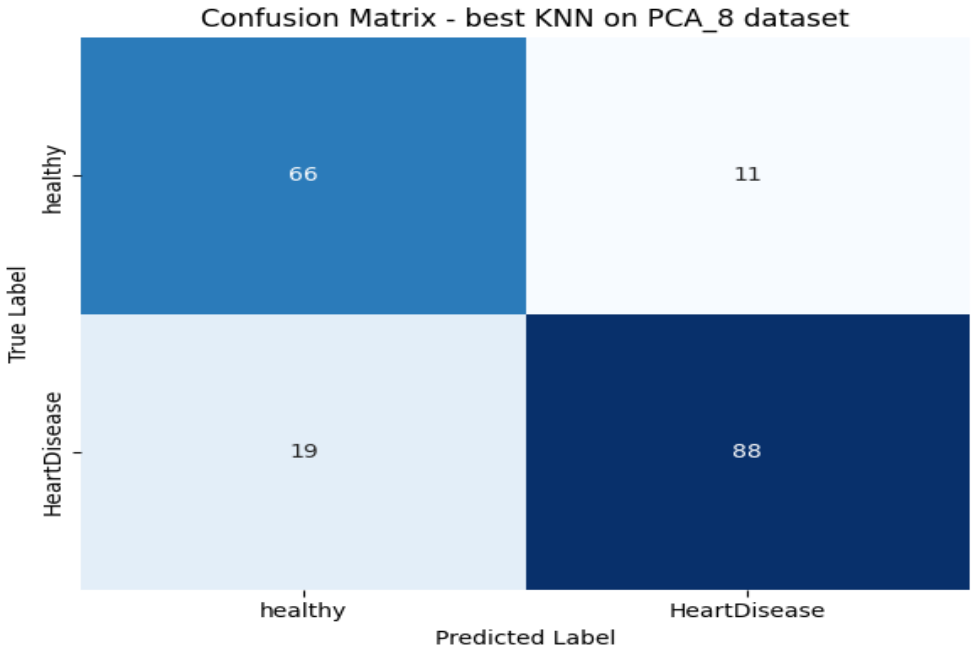
Rysunek 9: Najlepszy KNN na zbiorze przeskalowanym



Rysunek 10: Najlepszy KNN na zbiorze PCA 4

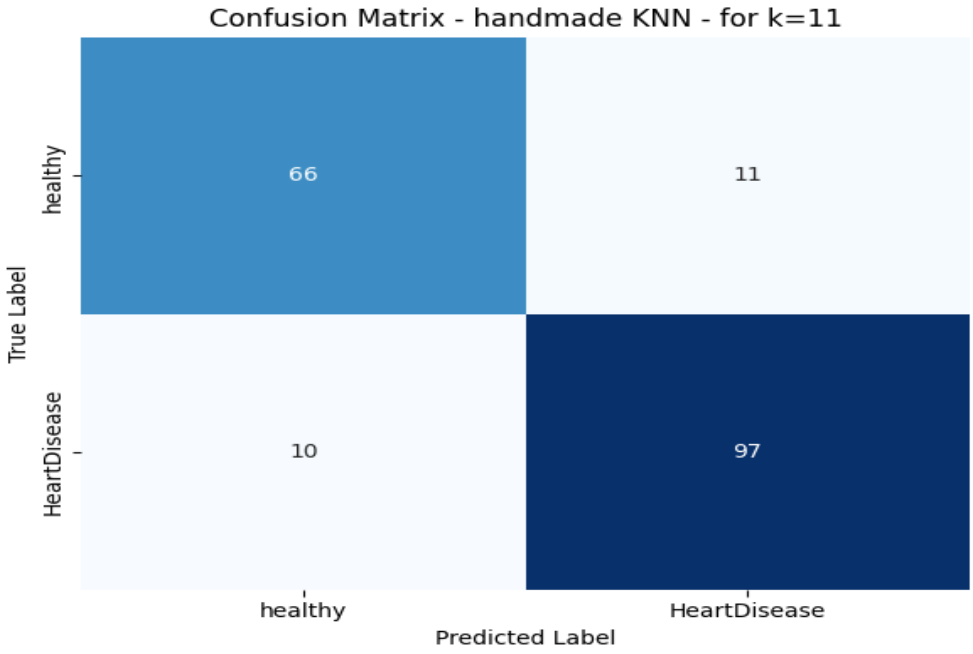


Rysunek 11: Najlepszy KNN na zbiorze PCA 6

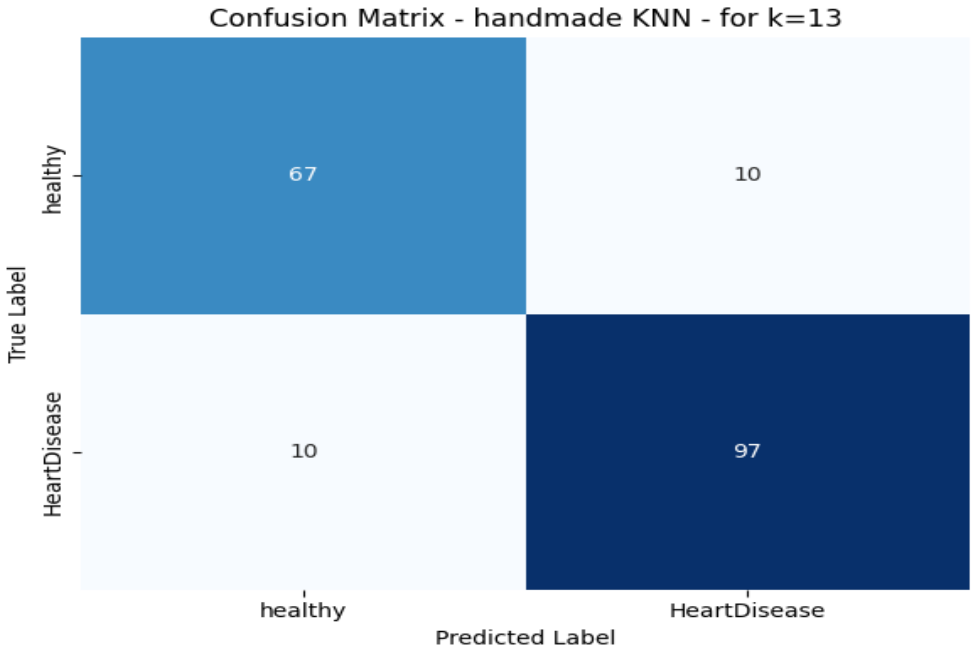


Rysunek 12: Najlepszy KNN na zbiorze PCA 8

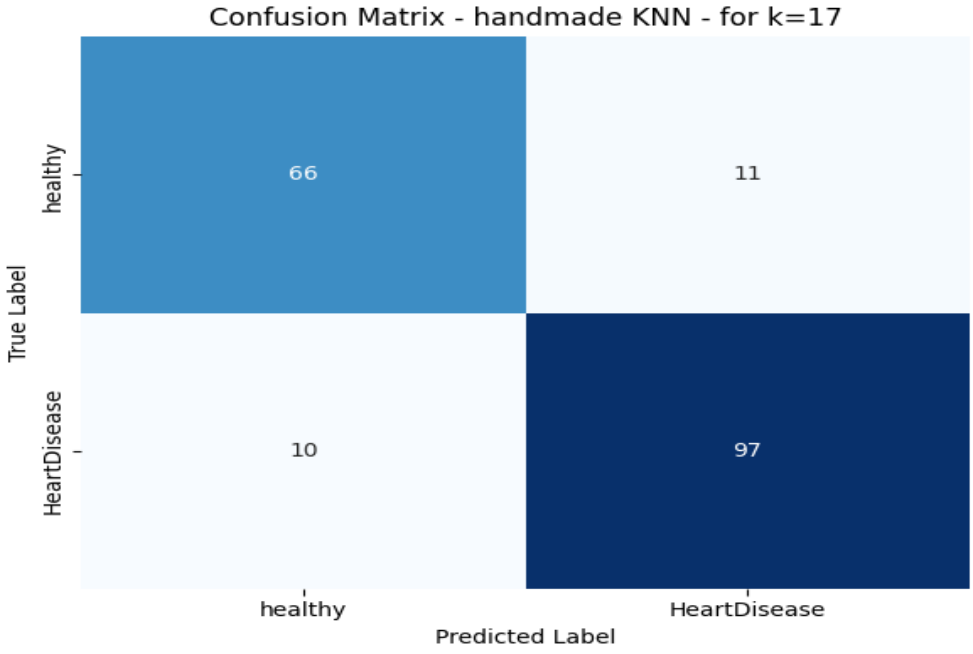
Wyniki naszego KNN-a na zbiorze przeskalowanym, bo wcześniej na nim uzyskaliśmy najlepsze parametry. (najlepsze wyniki,co ciekawe, dla k=13 i k=17) (Wcześniej przy grid searchu najlepsze wyniki dla k=11)



Rysunek 13: Nasz KNN na zbiorze przeskalowanym dla k=11

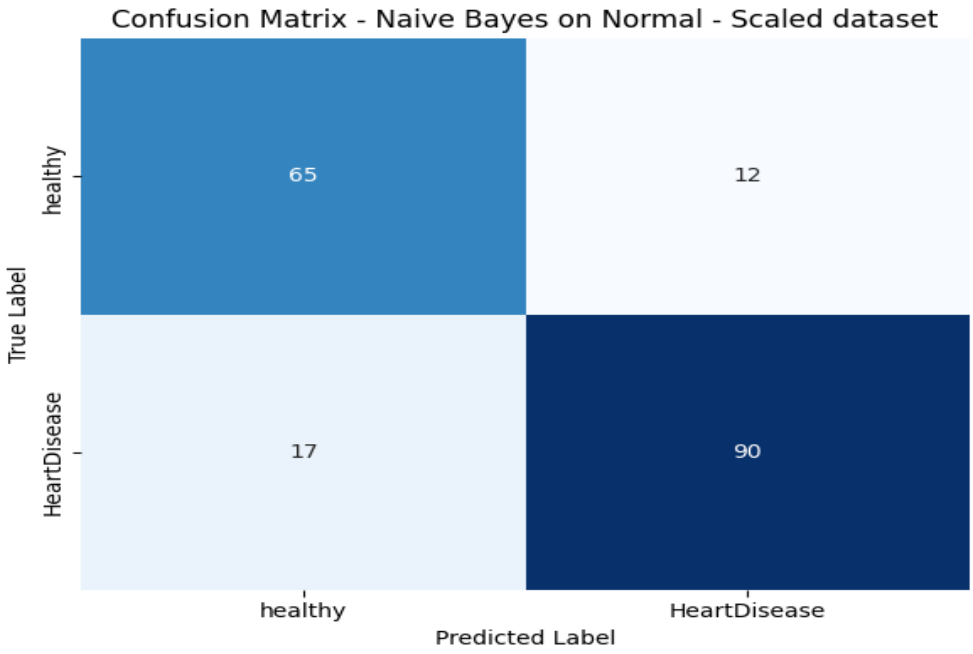


Rysunek 14: Nasz KNN na zbiorze przeskalowanym dla k=13

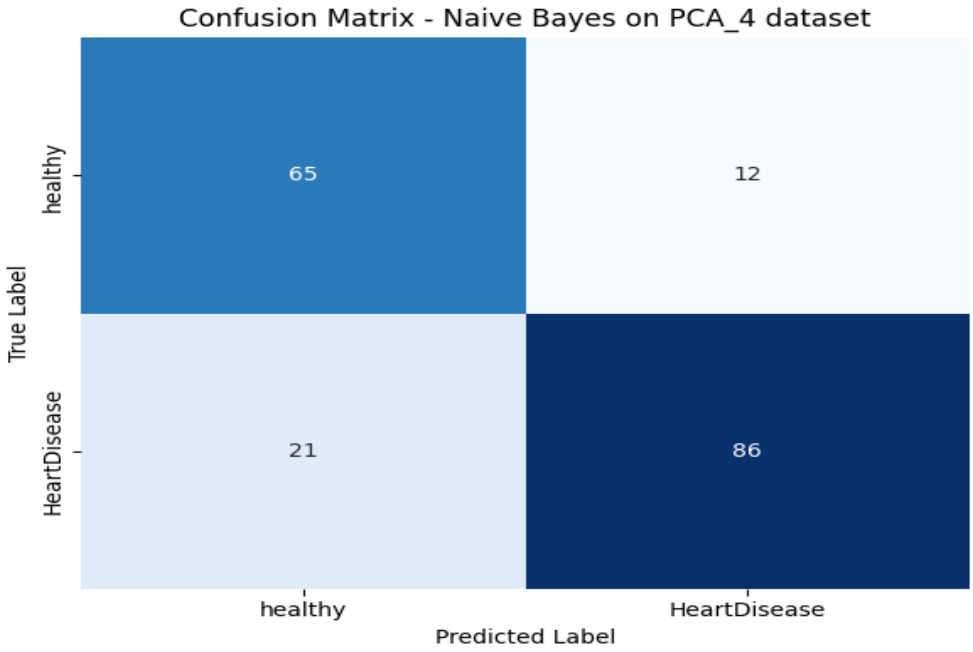


Rysunek 15: Nasz KNN na zbiorze przeskalowanym dla k=17

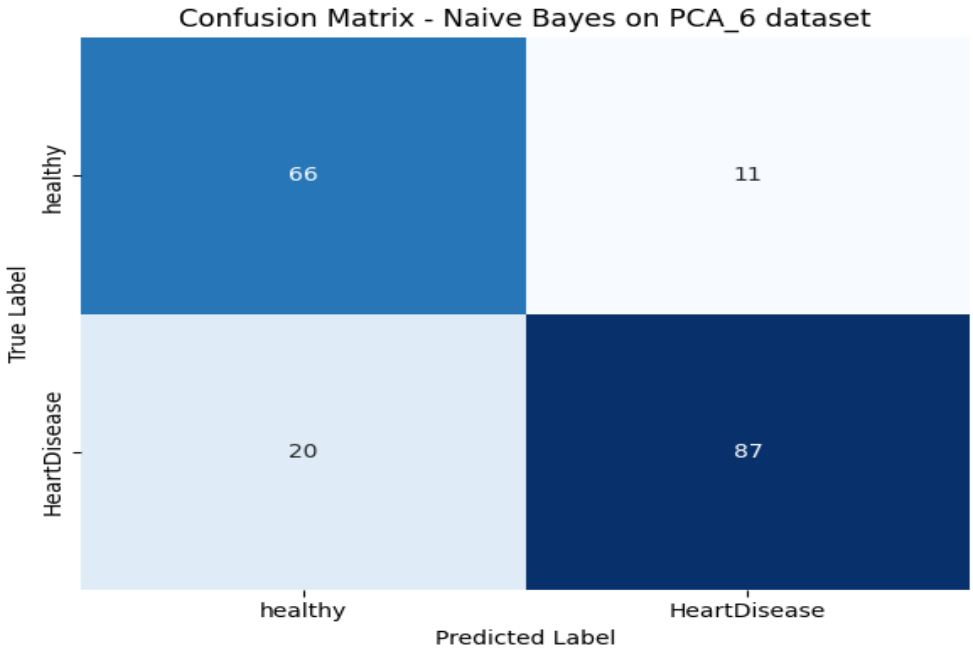
Wyniki Naive Bayes z sklearn-a w zależności od zbioru danych:



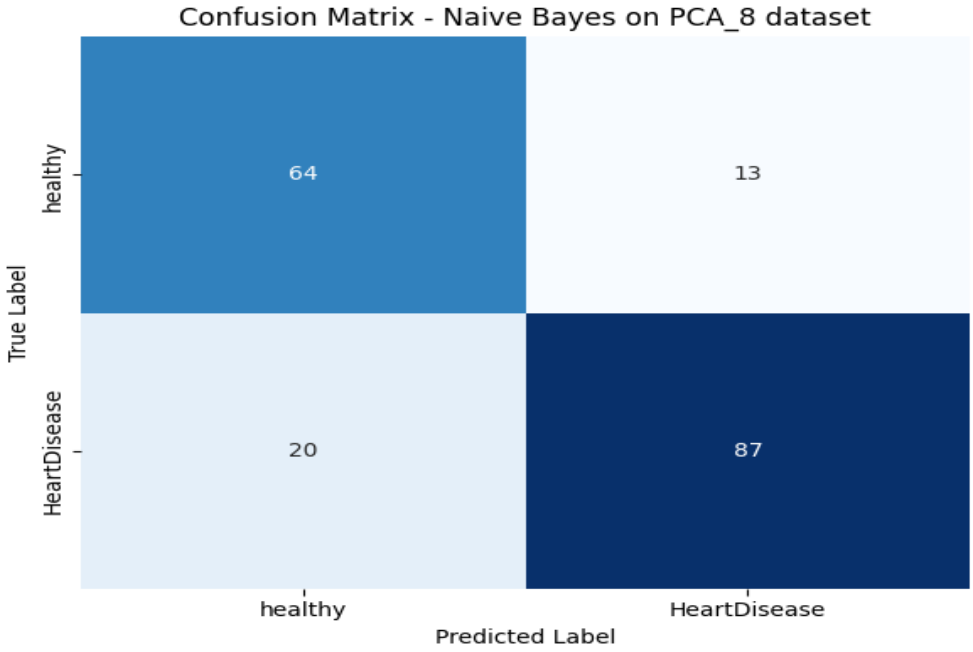
Rysunek 16: Naiwny Bayes na zbiorze przeskalowanym



Rysunek 17: Naiwny Bayes na zbiorze PCA 4

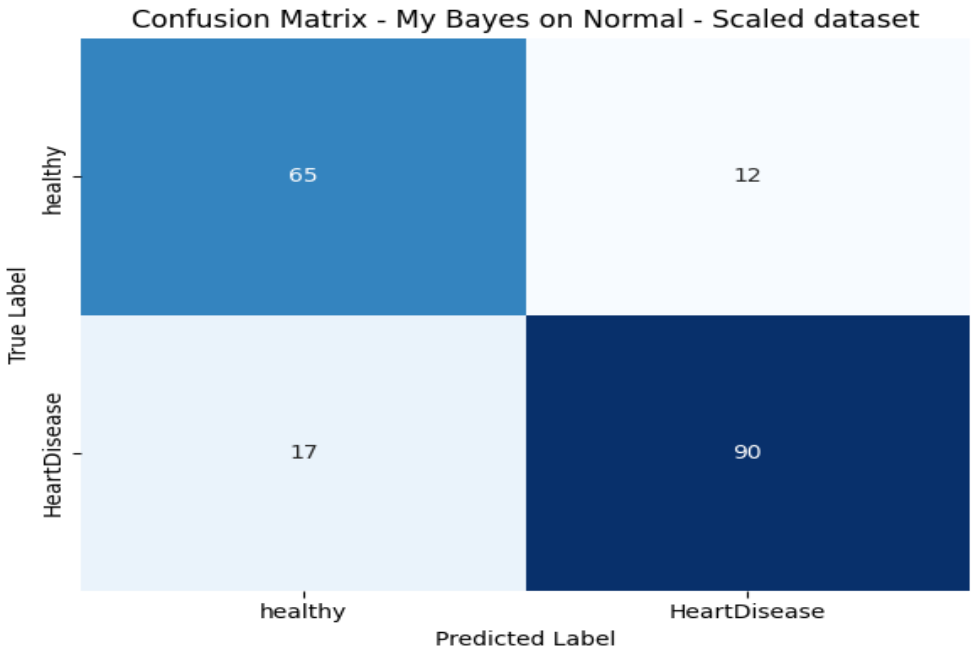


Rysunek 18: Naiwny Bayes na zbiorze PCA 6



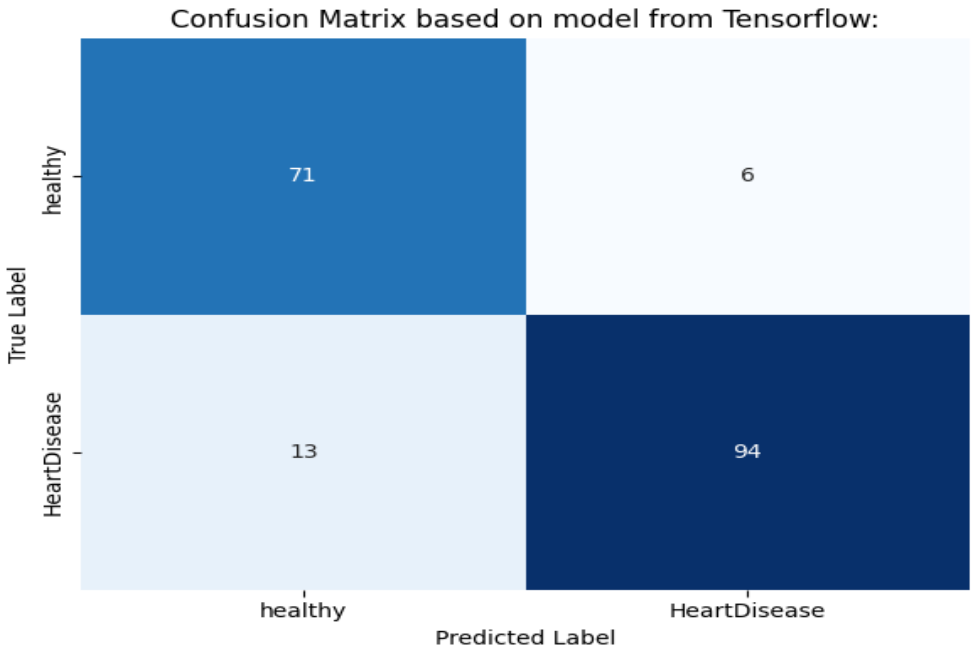
Rysunek 19: Naiwny Bayes na zbiorze PCA 8

Wyniki naszego klasyfikatora Bayesa były dokładnie takie same na każdym zbiorze jak wyniki Naiwnego Bayesa z sk-lerna:

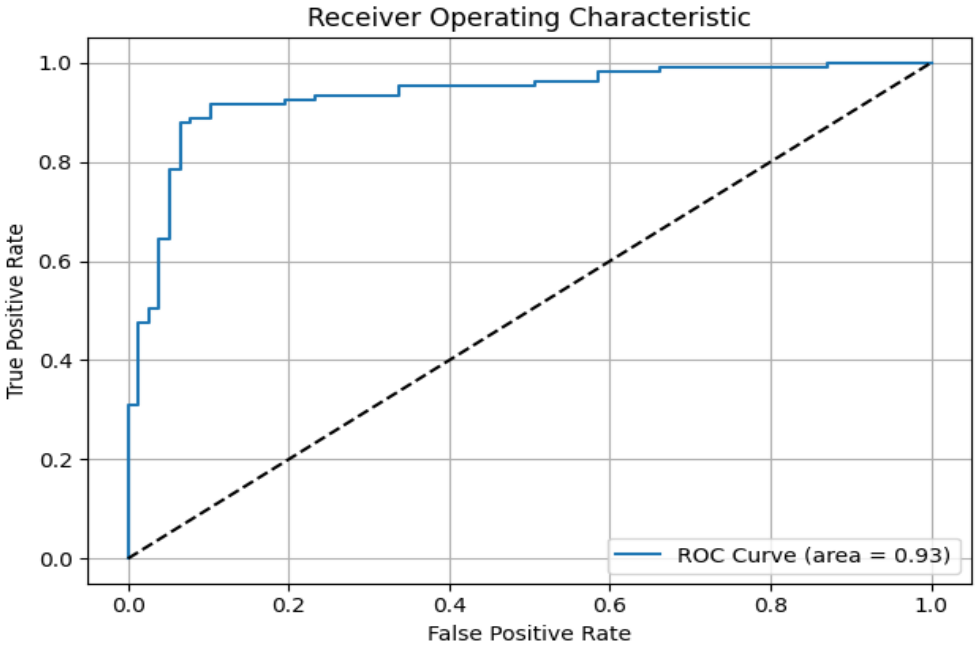


Rysunek 20: Nasz Bayes na zbiorze przeskalowanym

Niżej dodane są parametry które uzyskała prosta sieć neuronowa:



Rysunek 21: Prosta sieć neuronowa z TensorFlow



Rysunek 22: Wykres dla prostej sieci neuronowej z TensorFlow

Jeśli natomiast chodzi o metryki (accuracy i recall - ważna pod kątem medycznym) to najlepsze parametry uzyskaliśmy dla naszego KNN, następnie dla KNN z sklearna, później dla sieci neuronowej, a najsłabsze dla Naiwnych Bayesów, co potwierdziło nasze wstępne przypuszczenia.

```
FOR K=13
Accuracy:  0.8913043478260869
Classification Report:

              precision    recall  f1-score   support

           0       0.87      0.87      0.87        77
           1       0.91      0.91      0.91       107

    accuracy                           0.89       184
   macro avg       0.89      0.89      0.89       184
weighted avg       0.89      0.89      0.89       184
```

Rysunek 23: Wyniki dla naszego KNN dla k=13 na zbiorze przesalowanym

```
Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.92      0.88        77
           1       0.94      0.88      0.91       107

    accuracy                           0.90       184
   macro avg       0.89      0.90      0.90       184
weighted avg       0.90      0.90      0.90       184

ROC AUC Score: 0.9343366913460371
```

Rysunek 24: Wyniki dla prostej sieci neuronowej

```
-----MY Naive BAYES, DATASET:  Normal - Scaled

TEST ACCURACY: 0.842391304347826

Classification Report:

              precision    recall  f1-score   support

           0       0.79      0.84      0.82        77
           1       0.88      0.84      0.86       107

    accuracy                           0.84       184
   macro avg       0.84      0.84      0.84       184
weighted avg       0.84      0.84      0.84       184
```

Rysunek 25: Wyniki dla naszego klasyfikatora Bayesa na zbiorze przeskalowanym

Podsumowanie i wnioski

Z obserwacji wynika iż najlepsze parametry uzyskaliśmy dla naszego KNN, następnie dla KNN z sklearna, później dla sieci neuronowej, a najsłabsze dla Naiwnych Bayesów, co potwierdziło nasze wstępne przypuszczenia. Pomiomo iż algorytm KNN jest najbardziej złożony obliczeniowo to osiąga on bardzo dobre wyniki jeśli parametry są odpowiednio dobrane. Kolejną cenną obserwacją jest to, że redukcja wymiarowości nie wpłynęła na poprawę wyników klasyfikatorów, ponieważ najlepsze wyniki w każdym przypadku osiągaliśmy dla zbioru pełnego z przeskalowanymi danymi na wartości numeryczne.
Pomimo iż osiągamy wartości accuraccy i recall zbliżone do 90% to jednak zawsze pozostaną pewne przypadki nie wykryte dlatego bardzo ważne jest dbanie o swoje zdrowie i odpowiedni tryb życia.

Pełen kod aplikacji

Data_exploration.py:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

#Klasa do analizy danych. Wej ciowy dataset musi by  w postaci
    liczbowej. Mo na wywo a  summarize albo pojedyncze metody
class Data_exploration():
    def __init__(self,dataset):    #input=pandas.dataFrame
        self.data=dataset

    def info(self):
        print("Information about database:\n")
        print(self.data.info())

    def mising_values(self):
        print("Missing data:\n", self.data.isna().sum())

    def head(self):
        print("First 5 rows:\n", self.data.head())

    def describtion(self):
        print("Database description:\n", self.data.describe())

    def unique_values(self):
        for col in self.data.columns:
            print(f'Unique values in "{col}" column:\n', self.data[col].
                unique())

    def num_of_samples_per_class(self):
        print("Number of samples in each class:\n", self.data['
            HeartDisease'].value_counts())

    def histograms(self):
```

```
32          self.data.hist(figsize=(14, 8), bins=15)
33          plt.suptitle("Histograms for all features", fontsize=16)
34          plt.tight_layout()
35          plt.show()
36
37      def boxplots(self):
38          for col in self.data.columns:
39              if col != 'HeartDisease':
40                  plt.figure(figsize=(4, 2))
41                  sns.boxplot(data=self.data, x='HeartDisease', y=col)
42                  plt.title(f"{col} vs HeartDisease")
43                  plt.show()
44
45      def distributions(self):
46          for col in self.data.columns:
47              if col != 'HeartDisease':
48                  plt.figure(figsize=(8, 6))
49                  sns.histplot(data=self.data, x=col, hue='HeartDisease',
                        multiple='stack', kde=True)
50                  plt.title(f'Distribution of HeartDisease by {col}')
51                  plt.xlabel(f'{col}')
52                  plt.ylabel('Count')
53                  plt.show()
54
55      def pair_plots(self):
56          sns.pairplot(self.data, hue="HeartDisease", corner=True)
57          plt.suptitle("Scatterplots between features by HeartDisease")
58          plt.show()
59
60      def correlation_matrix(self):
61          plt.figure(figsize=(10, 8))
62          sns.heatmap(self.data.corr(), annot=True, cmap='coolwarm', fmt="
                .2f")
63          plt.title("Correlation Matrix")
64          plt.show()
65
66      def mean_values_per_class(self):
67          means = self.data.groupby("HeartDisease").mean()
68          means.T.plot(kind='bar', figsize=(14, 6))
69          plt.title("Mean values of features depending on the HeartDisease
                ")
70          plt.ylabel("Mean")
71          plt.xlabel("Features")
72          plt.xticks(rotation=45)
73          plt.legend(title="HeartDisease")
74          plt.tight_layout()
75          plt.show()
76
77
78      def summarize(self):
79          self.info()
80          self.mising_values()
81          self.head()
82          self.describtion()
83          self.unique_values()
```

```
        self.num_of_samples_per_class()
        self.histograms()
        self.distributions()
        self.pair_plots()
        self.correlation_matrix()
        self.mean_values_per_class()
```

KNN.py:

```
  # Hand-made KNN:
import math
import pandas as pd

class KNNClass:
    def metric(self, a, b):
        sum = 0
        if len(a) == len(b):

            # Canberra metric
            for i in range(len(a)):
                denom = math.fabs(a[i]) + math.fabs(b[i])
                if denom != 0:
                    sum += math.fabs(a[i] - b[i]) / denom
                else:
                    sum += 1

            return sum
        else:
            print("Different dimensions")

    def knn(self, X_train, Y_train, X_test, k):
        Y_prediction = []
        Y_train_np = Y_train.to_numpy()

        for X_test_vector in X_test:
            distances = []

            for i in range(len(X_train)):
                dist = self.metric(X_test_vector, X_train[i])
                distances.append((dist, Y_train_np[i]))

            distances.sort(key=lambda x: x[0])
            k_neighbors = distances[:k]  # list of tuples

            labels = [neighbor[1] for neighbor in k_neighbors]

            control = set(labels)
            most_common = ["", 0]
            for item in control:
                if (labels.count(item) > most_common[1]):
                    most_common = [item, labels.count(item)]

            Y_prediction.append(most_common[0])

```

```
46          return pd.Series(Y_prediction)
```

load_data.py:

```
1     # link: https://www.kaggle.com/datasets/fedesoriano/heart-failure-
          prediction
2
3 import pandas as pd
4 from sklearn.preprocessing import LabelEncoder
5
6 def load_heart_data(path="heart.csv"):
7     data = pd.read_csv(path)
8     return data
9
10 def data_to_int(data):
11     dataset = data.copy()
12     for col in dataset.columns:
13         if dataset[col].dtype == 'object':
14             le = LabelEncoder()
15             dataset[col] = le.fit_transform(dataset[col])
16     return dataset
```

model_neuron.py:

```
1 from tensorflow.keras.models import Sequential
2 from tensorflow.keras.layers import Dense
3
4 def create_nn_model(input_shape):
5     model = Sequential([
6         Dense(64, activation='relu', input_shape=(input_shape,)),
7         Dense(32, activation='relu'),
8         Dense(1, activation='sigmoid')
9     ])
10
11     model.compile(
12         optimizer='adam',
13         loss='binary_crossentropy',
14         metrics=['accuracy']
15     )
16     return model
```

Naive_Bayes.py:

```
1     # Hand-made Naive Bayes
2 from math import sqrt, pi, exp
3 import numpy as np
4 import pandas as pd
5
6 class NaiveBayes:
7
8     def gaussian(self, x, mean, std):
9         if std == 0:
```

```
            return 1.0 if x == mean else 0.0
        exponent = exp(-((x - mean) ** 2.0) / (2.0 * std ** 2.0))
        return (1 / (sqrt(2.0 * pi) * std)) * exponent

    def summarize_by_class(self, X, y):
        summaries = {}
        for cls in np.unique(y):
            X_cls = X[y == cls]
            means = X_cls.mean(axis=0)
            stds = X_cls.std(axis=0)
            summaries[cls] = list(zip(means, stds))
        return summaries

    def calculate_class_probabilities(self, summaries, input_vector,
       class_priors):
        probabilities = {}
        for cls, feature_summaries in summaries.items():
            prob = class_priors[cls]
            for i in range(len(input_vector)):
                mean, std = feature_summaries[i]
                prob *= self.gaussian(input_vector[i], mean, std)  # P(
                   Xi|Class)
            probabilities[cls] = prob
        return probabilities

    def predict(self, summaries, input_vector, class_priors):
        probs = self.calculate_class_probabilities(summaries,
           input_vector, class_priors)  # dictionary cls: prob
        return max(probs, key=probs.get)

    def get_predictions(self, X_train, Y_train, X_test):
        summaries = self.summarize_by_class(X_train, Y_train)
        class_priors = {cls: np.mean(Y_train == cls) for cls in np.
           unique(Y_train)}
        preds = [self.predict(summaries, row, class_priors) for row in
           X_test]
        return pd.Series(preds)
```

preprocess.py:

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder

def preprocess_data(data):

    for col in data.columns:
        if data[col].dtype == 'object':
            le = LabelEncoder()
            data[col] = le.fit_transform(data[col])

    X = data.drop("HeartDisease", axis=1)
    y = data["HeartDisease"]

```

```
    scaler = StandardScaler()
    X_scaled = scaler.fit_transform(X)

    X_train, X_test, y_train, y_test = train_test_split(
        X_scaled, y, test_size=0.2, random_state=42
    )

    return X_train, X_test, y_train, y_test
```

train.py:

```
from load_data import load_heart_data
from preprocess import preprocess_data
from model_neuron import create_nn_model

# Za adowanie i wst pne przetworzenie danych
data = load_heart_data("heart.csv")
X_train, X_test, y_train, y_test = preprocess_data(data)

# Stworzenie i trening modelu
model = create_nn_model(X_train.shape[1])
history = model.fit(
    X_train, y_train,
    epochs=50,
    batch_size=32,
    validation_split=0.1,
    verbose=1
)

# Zapisanie modelu i danych
model.save("heart_failure_nn_model.h5")
import joblib
joblib.dump((X_test, y_test), "test_data.pkl")
```

evaluate.py:

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score, classification_report,
    roc_auc_score, roc_curve, confusion_matrix
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
from KNN import *
from sklearn.naive_bayes import GaussianNB
from Naive_Bayes import *
import joblib
from tensorflow.keras.models import load_model
from Data_exploration import Data_exploration
from load_data import load_heart_data, data_to_int
from preprocess import preprocess_data

#Analiza danych
```

```
dataset = load_heart_data("heart.csv")
data_int = data_to_int(dataset)
data_exploration = Data_exploration(data_int)
data_exploration.summarize()

X_train, X_test, Y_train, Y_test = preprocess_data(dataset)

#Zbiory po redukcji wymiarowo ci
pca_4 = PCA(n_components=4)
pca_6 = PCA(n_components=6)
pca_8 = PCA(n_components=8)
X_train_pca_4 = pca_4.fit_transform(X_train)
X_test_pca_4 = pca_4.transform(X_test)
X_train_pca_6 = pca_6.fit_transform(X_train)
X_test_pca_6 = pca_6.transform(X_test)
X_train_pca_8 = pca_8.fit_transform(X_train)
X_test_pca_8 = pca_8.transform(X_test)


datasets = [
    ('Normal - Scaled', X_train, X_test, Y_train, Y_test),
    ('PCA_4', X_train_pca_4, X_test_pca_4, Y_train, Y_test),
    ('PCA_6', X_train_pca_6, X_test_pca_6, Y_train, Y_test),
    ('PCA_8', X_train_pca_8, X_test_pca_8, Y_train, Y_test)
]


#       -----KNN-----

param_grid = {
    'n_neighbors': [3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'canberra', 'minkowski']
}

#Szukanie odpowiednich parametr w za pomoc  GridSearcha
for name, X_tr, X_te, Y_tr, Y_te in datasets:
    grid_search = GridSearchCV(KNeighborsClassifier(), param_grid, cv=5,
        scoring='recall')
    grid_search.fit(X_tr, Y_tr)

    print(f'\n----- KNN on {name} dataset -----\n')
    print("Best parameters found: ", grid_search.best_params_)
    print("\nBest cross-validation accuracy:", grid_search.best_score_)

    best_knn = grid_search.best_estimator_
    Y_pred = best_knn.predict(X_te)

    print("\nTEST ACCURACY:", accuracy_score(Y_te, Y_pred))
    print('\nClassification Report:')
    print("\n", classification_report(Y_te, Y_pred))
    conf_mat = confusion_matrix(Y_te, Y_pred)
    class_names = ['healthy', 'HeartDisease']
    plt.figure(figsize=(6, 5))
    sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', cbar=False,
```

```
            xticklabels=class_names,
71                  yticklabels=class_names)
72      plt.xlabel('Predicted Label')
73      plt.ylabel('True Label')
74      plt.title(f'Confusion Matrix - best KNN on {name} dataset')
75      plt.tight_layout()
76      plt.show()
77
78
79  #Sprawdzenie w asnym KNN zbudowanym na bazie poprzednich wskaza
80  for neighbours in range(8,19):
81      my_knn = KNNClass()
82      Y_pred_handmade = my_knn.knn(X_train, Y_train, X_test, k=neighbours)
83      accuracy=accuracy_score(Y_test, Y_pred_handmade)
84      print(f"FOR K={neighbours}")
85      print("Accuracy: ",accuracy)
86      print("Classification Report:\n")
87      print(classification_report(Y_test, Y_pred_handmade))
88      conf_mat = confusion_matrix(Y_test, Y_pred_handmade)
89      class_names = ['healthy', 'HeartDisease']
90      plt.figure(figsize=(6, 5))
91      sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
92      plt.xlabel('Predicted Label')
93      plt.ylabel('True Label')
94      plt.title(f'Confusion Matrix - handmade KNN - for k={neighbours}')
95      plt.tight_layout()
96      plt.show()
97
98  #Najlepsze wyniki dla k=13 i k=17
99
100
101
102
103 #       -----NAIVE BAYES-----
104
105 #from sklearn
106 for name, X_tr, X_te, Y_tr, Y_te in datasets:
107     model_bayes = GaussianNB()
108     model_bayes.fit(X_tr, Y_tr)
109     Y_pred = model_bayes.predict(X_te)
110     print("-----NAIVE BAYES, DATASET: ",name)
111     print("\nTEST ACCURACY:", accuracy_score(Y_te, Y_pred))
112     print('\nClassification Report:')
113     print("\n",classification_report(Y_te, Y_pred))
114     conf_mat = confusion_matrix(Y_te, Y_pred)
115     class_names = ['healthy', 'HeartDisease']
116     plt.figure(figsize=(6, 5))
117     sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
118     plt.xlabel('Predicted Label')
119     plt.ylabel('True Label')
120     plt.title(f'Confusion Matrix - Naive Bayes on {name} dataset')
121     plt.tight_layout()
122     plt.show()
```

```
123
124
125 #Handmade Naive Bayes
126 for name, X_tr, X_te, Y_tr, Y_te in datasets:
127     my_bayes = NaiveBayes()
128     Y_pred_my_NB = my_bayes.get_predictions(X_tr, Y_tr, X_te)
129     print("-----MY Naive BAYES, DATASET: ",name)
130     print("\nTEST ACCURACY:", accuracy_score(Y_te, Y_pred_my_NB))
131     print('\nClassification Report:')
132     print("\n",classification_report(Y_te, Y_pred_my_NB))
133     conf_mat = confusion_matrix(Y_te, Y_pred_my_NB)
134     class_names = ['healthy', 'HeartDisease']
135     plt.figure(figsize=(6, 5))
136     sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', cbar=False,
            xticklabels=class_names, yticklabels=class_names)
137     plt.xlabel('Predicted Label')
138     plt.ylabel('True Label')
139     plt.title(f'Confusion Matrix - My Bayes on {name} dataset')
140     plt.tight_layout()
141     plt.show()
142
143
144
145
146
147 #  ---- MODEL TENSORFLOW W CELU DOK ADNIEJSZEJ ANALIZY  ----
148
149 # Za adowanie modelu i danych testowych
150 model = load_model("heart_failure_nn_model.h5")
151 X_test_model, y_test_model = joblib.load("test_data.pkl")
152
153 # Przewidywanie i ocena
154 y_pred_probs_model = model.predict(X_test_model).flatten()
155 y_pred_model = np.round(y_pred_probs_model)
156
157 #Podsumowanie
158 print("\nClassification Report:")
159 print(classification_report(y_test_model, y_pred_model))
160 print("ROC AUC Score:", roc_auc_score(y_test_model, y_pred_probs_model))
161 print('\nConfusion Matrix:')
162 conf_mat = confusion_matrix(y_test_model, y_pred_model)
163 class_names = ['healthy', 'HeartDisease']
164 plt.figure(figsize=(6, 5))
165 sns.heatmap(conf_mat, annot=True, fmt='d', cmap='Blues', cbar=False,
       xticklabels=class_names, yticklabels=class_names)
166 plt.xlabel('Predicted Label')
167 plt.ylabel('True Label')
168 plt.title('Confusion Matrix based on model from Tensorflow:')
169 plt.tight_layout()
170 plt.show()
171
172 # WYkres
173 fpr, tpr, thresholds = roc_curve(y_test_model, y_pred_probs_model)
174 plt.figure()
175 plt.plot(fpr, tpr, label='ROC Curve (area = {:.2f})'.format(
```

```
    roc_auc_score(y_test_model, y_pred_probs_model)))
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc='lower right')
plt.grid(True)
plt.tight_layout()
plt.savefig("roc_curve.png")
plt.show()
```

Bibliografia

- Python Data Science, Jake VanderPlas
- Data Science od podstaw, Joel Grus
- Czyszczenie danych w Pythonie. Receptury, Michael Walker
- Wikipedia