Dokumentacja projektu z przedmiotu Sztuczna inteligencja

Predykcja otyłości z wykorzystaniem sieci neuronowych

Autor:
Katarzyna Pieczonka
nr albumu: 132785

Prowadzący: mgr inż. Wojciech Gałka
Kierunek: Informatyka, II stopień
Semestr: letni
Rok akademicki: 2023/2024

1. Ogólne założenia projektu

Ogólnym celem projektu jest porównanie różnych modeli uczenia maszynowego oraz zbudowanie sieci neuronowej zdolnej do predykcji na podstawie podanych przez użytkownika danych, czy ma on prawidłową, czy nieprawidłową wagę, osiągając jak najwyższy możliwy procent poprawnych predykcji. Model ma pomóc w klasyfikacji pacjentów na podstawie ustalonego zestawu cech.

Aplikacja wykorzystuje do uczenia modelu ogólnodostępnego zbioru danych “Obesity Risk Dataset”. Zbiór dzielony jest na część treningową i testową, następnie wykonywane jest uczelnie modelu, oraz krosswalidacja. Użytkownik podaje dane - odpowiadając na pytania - dla których wykonywana jest predykcja i wypisywany jest wynik.

1. Zbiór danych

Zbiór danych na których aplikacja wykonuje uczenie modelu jest zbiór “Obesity Risk Dataset”. Jest to zbiór ogólnodostępny i darmowy, który można pobrać z poniższych stron:

https://www.kaggle.com/datasets/jpkochar/obesity-risk-dataset

lub

https://archive.ics.uci.edu/dataset/544/estimation+of+obesity+levels+based+on+eating+habits+and+physical+condition

Zbiór danych zawiera dane dotyczące otyłości wśród obywatelu Mekyku, Peru i Kolumbii, określanej na podstawie ich nawyków żywieniowych oraz kondycji fizycznej. Posiada on 2111 instancji danych, oraz 16 cech.

Opis cech na podstawie angielskiej wersji dokumentacji do zbioru danych:

1. **Płeć** - cecha kategoryczna (Mężczyzna/Kobieta) - w zbiorze znajduje się 50% danych dotyczących kobiet i 50% dotyczących mężczyzn
2. **Wiek** - cecha numeryczna - w zbiorze znajdują się dane dotyczące osób w wieku od 14 do 61 lat

3. **Wzrost** - cecha numeryczna - w zbiorze znajdują się dane dotyczące osób wzrostu od 1.45 m do 1.98 m
4. **Waga** - cecha numeryczna - w zbiorze znajdują się dane dotyczące osób posiadających wagę od 39 kg do 165 kg
5. **Historia otyłości w rodzinie** - cecha kategoryczna, binarna (yes/no - tak/nie)
6. **FAVC** (częste spożycie wysokokalorycznego jedzenia) - cecha kategoryczna, binarna (yes/no - tak/nie)
7. **FCVC** (występowanie warzyw w posiłkach) - cecha numeryczna w skali 1 do 3 (1 - rzadko, 3 - często)
8. **NCP** (ilość posiłków dziennie) - cecha numeryczna w skali od 1 do 4 (1 - 1-2 posiłki dziennie, 4 - 6-7 posiłków)
9. **CAEC** (spożycie jedzenia pomiędzy posiłkami) - cecha kategoryczna (0 - nigdy, sometimes - czasami, frequently - często, always - zawsze)
10. **SMOKE** (palenie papierosów) - cecha kategoryczna, binarna (yes/no - tak/nie)
11. **CH2O** (ilość wody dziennie) - cecha numeryczna w skali 1 do 3
12. **SCC** (monitorowanie kalorii) - cecha kategoryczna, binarna (yes/no - tak/nie)
13. **FAF** (aktywność fizyczna - jak często) - cecha numeryczna w skali 0 do 3
14. **TUE** (czas spędzony przed ekranem urządzeń elektronicznych - cecha numeryczna w skali 0 do 2
15. **CALC** (spożycie alkoholu - jak często) - cecha kategoryczna (0 - nigdy, sometimes - czasami, frequently - często, always - zawsze)
16. **MTRANS** (zazwyczaj używany środek transportu) - cecha kategoryczna, do wyboru: Public_Transportation (transport publiczny), Walking (chodzenie pieszo), Automobile (samochód), Motorbike (motocykl)

Klasa docelowa (target class):
NObeyesdad (poziom otyłości) - cecha kategoryczna - wynik predykcji w skali:

- Insufficient Weight - niedowaga
- Normal Weight - waga prawidłowa
- Overweight Level I - nadwaga typu I
- Overweight Level II - nadwaga typu II
- Obesity Type I - otyłość typu I
- Obesity Type II - otyłość typu II
- Obesity Type III - otyłość typu III

Zbiór danych nie posiada brakujących wartości, co sprawdzane jest również w programie:

```
# Sprawdzenie braków danych
missing_values = data.isnull().sum()
print("Braki danych:")
print(missing_values)
```

Wynik:

```
Braki danych:
Gender                              0
Age                                 0
Height                              0
Weight                              0
family_history_with_overweight      0
FAVC                                0
FCVC                                0
NCP                                 0
CAEC                                0
SMOKE                               0
CH2O                                0
SCC                                 0
FAF                                 0
TUE                                 0
CALC                                0
MTRANS                              0
NObeyesdad                          0
dtype: int64
```

3. Przekształcenia zbioru danych:

1. Sprawdzanie i usuwanie duplikatów

```python
#Sprawdzanie duplikatów
duplicates = dc.duplicated().sum()
print(f"Duplikaty: {duplicates}")
dc = dc.drop_duplicates()
print(f"Duplikaty po usunięciu: {dc.duplicated().sum()}")
```

Duplikaty sprawdzane są za pomocą metody `duplicated().sum()` z biblioteki Pandas. Funkcja ta liczy liczbę duplikatów, uwzględniając kolumnę z decyzją NObeyesdad. Jeśli funkcja napotyka identyczne dane, usuwa je za pomocą funkcji `drop_duplicates()`. Następnie występowanie duplikatów sprawdzane jest ponownie.

2. Kodowanie danych kategorycznych

Zbiór danych dzielony jest na kolumny kategoryczne oraz numeryczne.

```python
#Kolumny kategoryczne i numeryczne
categorical_cols = []
numeric_cols = []

for col in dc.columns:
    if pd.api.types.is_numeric_dtype(dc[col]):
        numeric_cols.append(col)
    elif pd.api.types.is_object_dtype(dc[col]):
        categorical_cols.append(col)
```

```
categorical_features = categorical_cols[:-1] #Kolumny kategoryczne bez kolumny z decyzją
```

Następnie kolumny kategoryczne są kodowane za pomocą `LabelEncoder` z biblioteki `sklearn.preprocessing`. Dla każdej z cech tworzona jest osobna instancja klasy `LabelEncoder`.

Kodowanie kolumny z decyzją. Wynik zapisywany jest do nowej kolumny:

```
label_encoder_target = LabelEncoder()
dc['NObeyesdad_Encoded'] = label_encoder_target.fit_transform(dc['NObeyesdad'])
dc['NObeyesdad_Encoded'].value_counts()
```

Kodowanie cech. Wyniki nadpisują dane w tych kolumnach:

```
label_encoders = {}

for col in categorical_features:
    le = LabelEncoder()
    dc[col] = le.fit_transform(dc[col])
    label_encoders[col] = le
```

3. Skalowanie danych

Dane skalowane są za pomocą klasy `StandardScaler` z biblioteki `sklearn.preprocessing`

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

4. Biblioteki

Biblioteki używane w projekcie:
- **tensorflow** - otwartoźródłowa biblioteka uczenia maszynowego i głębokiego uczenia. Umożliwia budowanie i trenowanie modeli sztucznej inteligencji, w tym sieci neuronowych, zarówno dla zadań klasyfikacyjnych, jak i regresyjnych.
- **keras** - biblioteka w języku Python, która zapewnia wysoką abstrakcję warstw, co pozwala na łatwe skonfigurowanie i użycie modeli uczenia maszynowego
- **pandas** - biblioteka do analizy danych w języku Python, która oferuje struktury danych i narzędzia do manipulacji tabelami (DataFrames). Pandas ułatwia wczytywanie, przetwarzanie i analizowanie danych
- **numpy** - biblioteka do obliczeń naukowych w Pythonie, oferująca wsparcie dla wielowymiarowych tablic oraz funkcje matematyczne i statystyczne. Jest podstawą dla wielu innych bibliotek analizy danych

- **scikit-learn (sklearn)** - biblioteka uczenia maszynowego w Pythonie, która oferuje różnorodne algorytmy do klasyfikacji, regresji, klasteryzacji i redukcji wymiarów, a także narzędzia do przetwarzania wstępnego danych i ewaluacji modeli.
- **joblib** - biblioteka w Pythonie, która jest wykorzystywana głównie do efektywnego wykonywania obliczeń, równoległego przetwarzania oraz serializacji (zapisywania i ładowania) obiektów.

5. Pliki w projekcie

- **model.py** - Plik zawiera model sieci neuronowej. Dane w zbiorze są najpierw przekształcane, potem dokonuje się uczenie sieci neuronowej i określane są miary jakości eksperymentu. Następnie gotowy model wraz z skalerem i encoderami zapisywane są do zewnętrznych plików.
- **model_test.py** - Plik zawiera porównanie różnych metod klasyfikacji i sieci neuronowej wraz z ich dokładnością.
- **prediction.py** - Plik umożliwiający pobieranie danych od użytkownika, na których następnie dokonywana jest predykcja i wypisywana decyzja za pomocą nauczonego wcześniej modelu sieci neuronowej

Pliki zawierającej obiekty używane do predykcji:
(label_encoder_target.joblib, label_encoders.joblib, model.keras, scaler.joblib)

Pliki te są następnie wczytywane podczas dokonywania predykcji:

```
# Wczytanie modelu i skalera
scaler = joblib.load('scaler.joblib')
label_encoders = joblib.load('label_encoders.joblib')
label_encoder_target = joblib.load('label_encoder_target.joblib')
model = load_model('model.keras')
```

6. Struktura sieci neuronowej

Sieć neuronowa w aplikacji zbudowana jest za pomocą bibliotek TensorFlow oraz Keras. Sieć ma następującą strukturę:

- warstwa wejściowa - warstwa ma taki rozmiar, ile cech wejściowych, czyli kolumn w macierzy danych wejściowych, jest w zbiorze X
- pierwsza warstwa ukryta - posiada 64 neurony oraz funkcję aktywacji ReLU
- druga warstwa ukryta - również posiada 64 neurony oraz funkcję aktywacji ReLU
- warstwa wyjściowa - liczba neuronów jest równa liczbie klas. Posiada funkcję aktywacji softmax.

Sieć trenowana była przez 500 epok ze współczynnikiem uczenia 0,001, oraz rozmiarem batcha (ilości danych przekazywanych w jednej porcji do sieci) 32.

```
def create_model(input_size, hidden_size, output_size):
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(hidden_size, input_shape=(input_size,), activation='relu'),
        tf.keras.layers.Dense(hidden_size, activation='relu'),
        tf.keras.layers.Dense(output_size, activation='softmax')
    ])
    return model
```

```python
# Parametry sieci
input_size = X.shape[1]
hidden_size = 64
output_size = len(np.unique(y))  # Liczba klas
num_epochs = 500
batch_size = 32
learning_rate = 0.001
```

7. Metoda kroswalidacji

W celu oceny wydajności modelu stosowana jest **5-krotna kroswalidacja stratyfikowana**. Dane w zbiorze podzielone są na 5 równych części, każda z nich zawierająca proporcjonalną reprezentację każdej klasy. Następnie model jest trenowany na 4 z tych części i testowany na piątej. Proces ten powtarzany jest 5 razy, gdzie za każdym razem jako zbiór testowy używana jest inna część zbioru.

```python
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
```

8. Miary jakości

Do oceny modelu stosowane są następujące miary

- dokładność (accuracy) - procent poprawnych predykcji spośród wszystkich predykcji

```python
accuracy_score(y_test_classes, y_pred_classes))
```

- F1-score - miara, która uwzględnia zarówno precyzję, jak i czułość - jest to średnia harmoniczna tych dwóch wartości

```python
f1_score(y_test_classes, y_pred_classes, average='weighted')
```

- precyzja (precision)

```python
precision_score(y_test_classes, y_pred_classes,
average='weighted')
```

- czułość (recall)

```python
recall_score(y_test_classes, y_pred_classes, average='weighted')
```

Ponadto, w projekcie zastosowana jest macierz pomyłek (Confusion Matrix), która pozwala zwizualizować wydajność algorytmu klasyfikacyjnego graficznie. Każda kolumna macierzy reprezentuje przewidywaną klasę, natomiast wiersz rzeczywisty wynik. Elementy na głównej przekątnej reprezentują liczbę poprawnej predykcji, a elementy poza główną przekątną liczbę błędnych predykcji.

9. Wynik eksperymentu

Celem eksperymentu było osiągnięcie jak najwyższego możliwego procentu poprawnych predykcji oraz zastosowanie nauczonej sieci neuronowej do przewidywania decyzji na podstawie danych podanych przez użytkownika.

Wyniki:

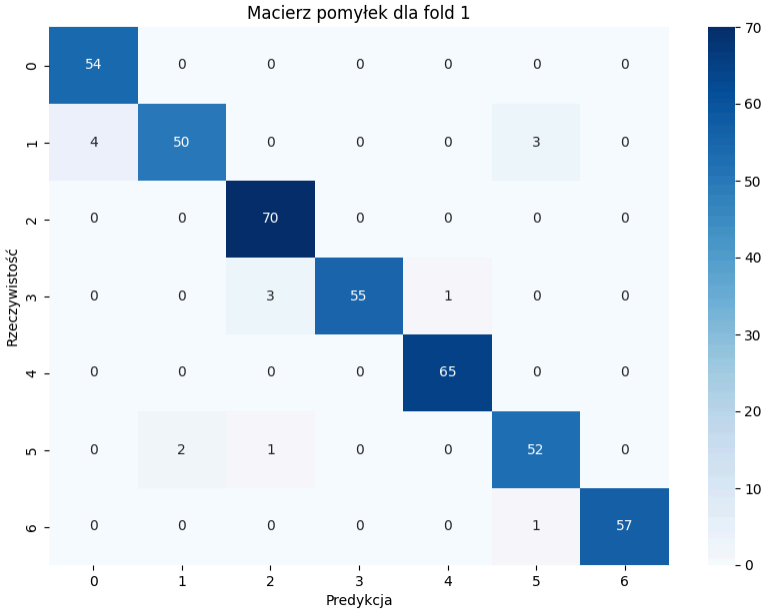
Dla 200 epok:

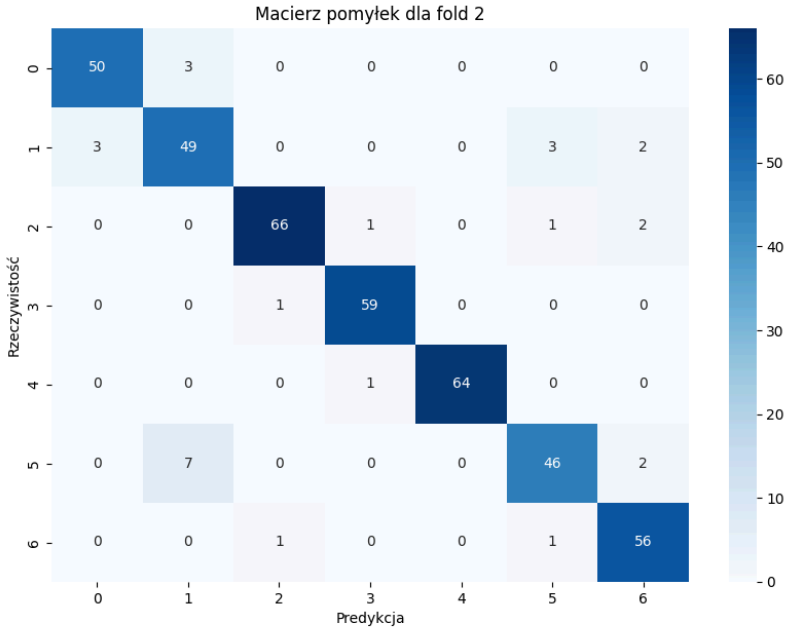
```
Średnia dokładność: 0.9483 (± 0.0190)
Średnia F1: 0.9483 (± 0.0188)
Średnia precyzja: 0.9493 (± 0.0184)
Średnia czułość: 0.9483 (± 0.0190)
```

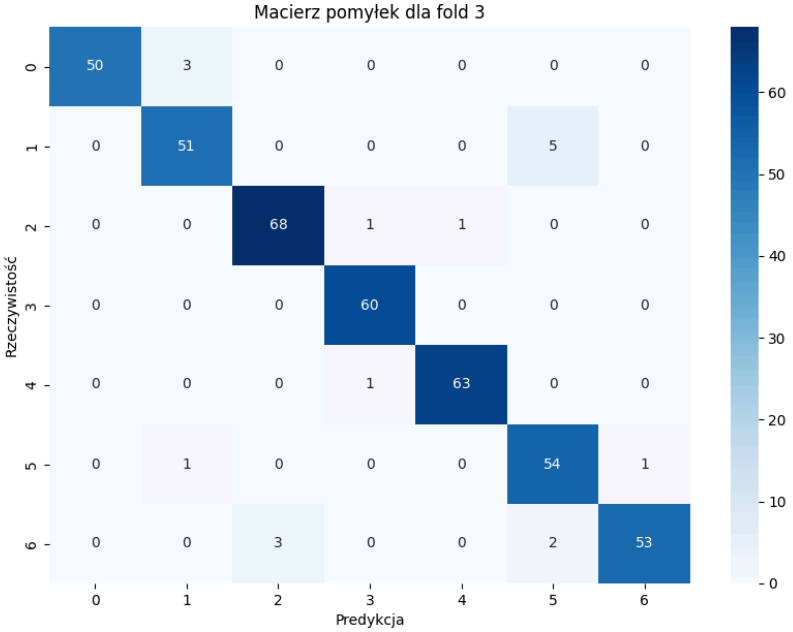
Dla 500 epok:

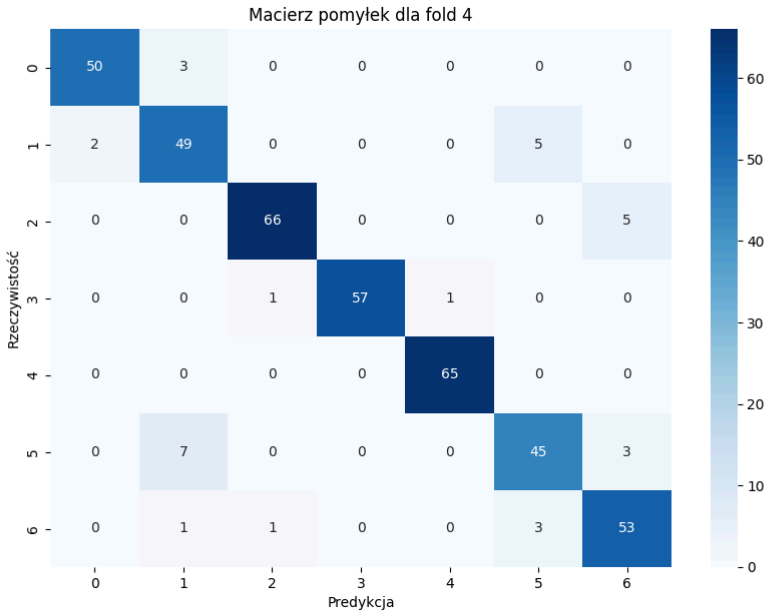
```
Średnia dokładność: 0.9506 (± 0.0105)
Średnia F1: 0.9508 (± 0.0104)
Średnia precyzja: 0.9515 (± 0.0105)
Średnia czułość: 0.9506 (± 0.0105)
```

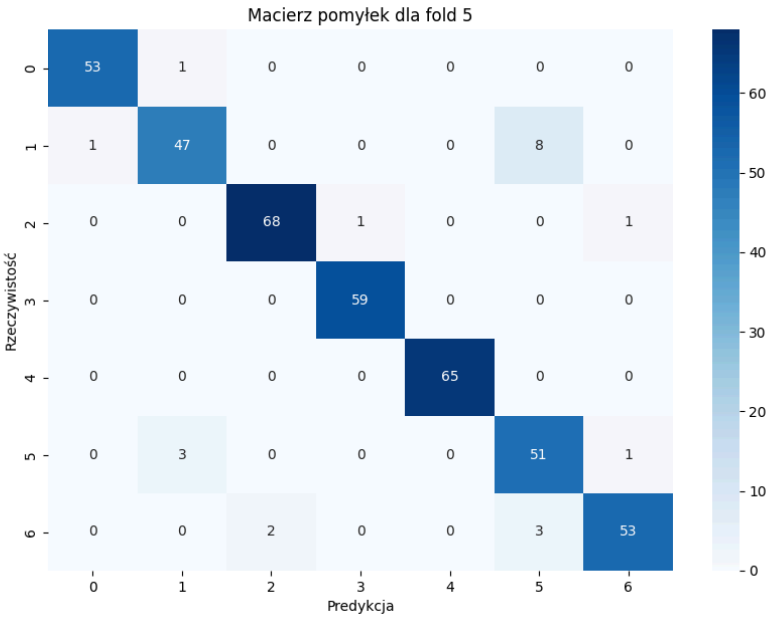
Macierze pomyłek dla poszczególnych foldów:




Macierz pomyłek dla fold 2
Rzeczywistość
Predykcja
0 1 2 3 4 5 6
50 3 0 0 0 0 0
3 49 0 0 0 3 2
0 0 66 1 0 1 2
0 0 1 59 0 0 0
0 0 0 1 64 0 0
0 7 0 0 0 46 2
0 0 1 0 0 1 56
0 10 20 30 40 50 60


Macierz pomyłek dla fold 3
Rzeczywistość
Predykcja
0 1 2 3 4 5 6
50 3 0 0 0 0 0
0 51 0 0 0 5 0
0 0 68 1 1 0 0
0 0 0 60 0 0 0
0 0 0 1 63 0 0
0 1 0 0 0 54 1
0 0 3 0 0 2 53
0 10 20 30 40 50 60





Wnioski: Model osiągnął wysoką dokładność i dobre wyniki. Być może zwiększenie liczby epok wpłynęłoby na zwiększenie dokładności, jednak byłoby to kosztem wydajności i czasu uczenia modelu, który mógłby znacznie wzrosnąć.

Eksperyment wykazał, że sieć neuronowa może skutecznie klasyfikować poziomy otyłości na podstawie dostarczonych cech.