



Uniwersytet
Bielsko-Bialski

Prognozowanie parametrów zdrowotnych pacjentów

Zadanie Ensemble

Sprawozdanie z Ćwiczeń
Nauka o Danych II

Data wykonania:
28.06.2025

Autor:
Bartosz Bieniek 058085

1. Cel ćwiczenia

Zadanie polega na prowadzeniu na własnym zbiorze danych (z poprzedniego zajęcia) kolejnych działań:

1. Porównaj dokładności modeli: Random Forest, XGBoost i Stacking.
2. Przeprowadź tuning hiper parametrów dla modelu XGBoost.
3. Wprowadź nowy model do zestawu stackingowego (np. KNN lub DecisionTreeClassifier).
4. Przetestuj modele na innym zbiorze danych (np. Wine, Iris).
5. Przedstaw wyniki w formie wykresu słupkowego porównującego dokładność.

Zadania umieścić na [Github](#).

2. Przebieg ćwiczenia

1. Zaimportowano wymagane biblioteki

Załadowano biblioteki służące do analizy danych, budowy modeli ensemble, strojenia hiperparametrów oraz wizualizacji wyników. Uwzględniono modele Random Forest, XGBoost, Stacking oraz klasyczne regresory i klasyfikatory pomocnicze.

```
# Krok 1: Import bibliotek
import pandas as pd
import numpy as np

from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor, StackingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import LinearRegression
from sklearn.neighbors import KNeighborsRegressor
from xgboost import XGBRegressor
from sklearn.metrics import r2_score
import matplotlib.pyplot as plt
```

[6] ✓ 0.0s

Rys. 1. Zaimportowano wymagane biblioteki

2. Wczytano dane wejściowe

Z pliku `synthetic_health_data.csv` wczytano dane zawierające pięć zmiennych wejściowych (wiek, BMI, aktywność fizyczna, kalorie, sen) oraz trzy zmienne wyjściowe (cukier, ciśnienie skurczowe, ciśnienie rozkurczowe). Dane te reprezentowały problem regresji wielowymiarowej.

```

# Krok 2: Wczytanie danych
df = pd.read_csv("synthetic_health_data.csv")
X = df[["wiek", "BMI", "aktywnosc", "kalorie", "sen"]]
y = df[["cukier", "cisnienie_skurczowe", "cisnienie_rozkurczowe"]]

```

7] ✓ 0.0s

Rys. 2. Wczytano dane wejściowe

3. Podzielono dane na zbiory treningowy i testowy

Zastosowano funkcję `train_test_split`, aby losowo podzielić dane w proporcji 80:20. Zbiór treningowy posłużył do nauki modeli, a testowy do ich niezależnej oceny.

```

# Krok 3: Podział danych na treningowe i testowe
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

```

[8] ✓ 0.0s

Rys. 3. Podzielono dane na zbiory treningowy i testowy

4. Zdefiniowano bazowe modele regresyjne

Zainicjalizowano modele `RandomForestRegressor`, `XGBRegressor` oraz `StackingRegressor`, łączący pierwszy i drugi model jako estymatory bazowe z regresją liniową jako meta-estymatorem.

```

# Krok 4: Definicja modeli bazowych
rf = RandomForestRegressor(random_state=42)
xgb = XGBRegressor(random_state=42)
stack = StackingRegressor(
    estimators=[('rf', rf), ('xgb', xgb)],
    final_estimator=LinearRegression()
)

```

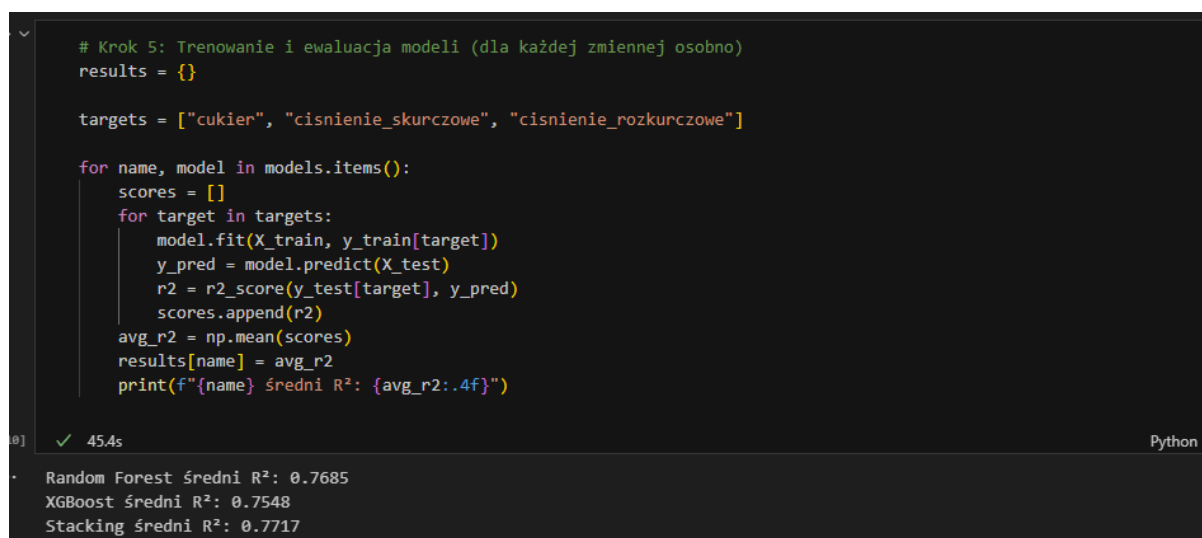
[9] ✓ 0.0s

Rys. 4. Zdefiniowano bazowe modele regresyjne

5. Przeprowadzono trenowanie i ocenę modeli

Dla każdego modelu przystąpiono do osobnego trenowania na każdej ze zmiennych wyjściowych. Obliczono współczynnik determinacji R^2 dla każdej

predykcji i uśredniono wyniki. Najwyższy średni R^2 uzyskał model stackingowy (0.7717), nieznacznie wyprzedzając Random Forest (0.7685) i XGBoost (0.7548).



```
# Krok 5: Trenowanie i ewaluacja modeli (dla każdej zmiennej osobno)
results = {}

targets = ["cukier", "cisnienie_skurczowe", "cisnienie_rozkurczowe"]

for name, model in models.items():
    scores = []
    for target in targets:
        model.fit(X_train, y_train[target])
        y_pred = model.predict(X_test)
        r2 = r2_score(y_test[target], y_pred)
        scores.append(r2)
    avg_r2 = np.mean(scores)
    results[name] = avg_r2
    print(f"{name} średni R²: {avg_r2:.4f}")
```

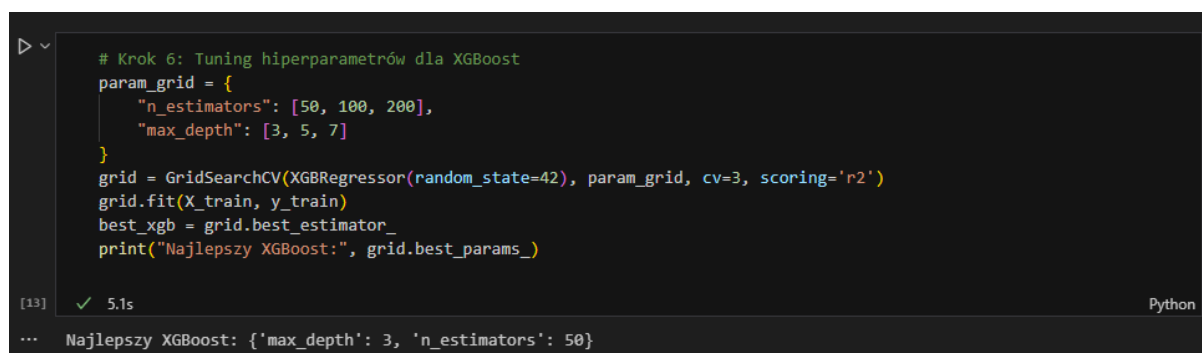
✓ 45.4s Python

Random Forest średni R^2 : 0.7685
XGBoost średni R^2 : 0.7548
Stacking średni R^2 : 0.7717

Rys. 5. Przeprowadzono trenowanie i ocenę modeli

6. Przeprowadzono tuning hiperparametrów modelu XGBoost

Zastosowano siatkę przeszukiwań (GridSearchCV) dla parametrów `max_depth` i `n_estimators`. Najlepszy zestaw parametrów to: `{'max_depth': 3, 'n_estimators': 50}`, który został wykorzystany w dalszych analizach.



```
# Krok 6: Tuning hiperparametrów dla XGBoost
param_grid = {
    "n_estimators": [50, 100, 200],
    "max_depth": [3, 5, 7]
}
grid = GridSearchCV(XGBRegressor(random_state=42), param_grid, cv=3, scoring='r2')
grid.fit(X_train, y_train)
best_xgb = grid.best_estimator_
print("Najlepszy XGBoost:", grid.best_params_)
```

[13] ✓ 5.1s Python

... Najlepszy XGBoost: {'max_depth': 3, 'n_estimators': 50}

Rys. 6. Przeprowadzono tuning hiperparametrów modelu XGBoost

7. Rozszerzono zestaw stackingowy o model KNN

Zbudowano nowy model stackingowy zawierający Random Forest, najlepszy XGBoost oraz KNN jako estymatory bazowe. W roli meta-estymatora

pozostawiono regresję liniową. Po uśrednieniu wyników R^2 dla trzech zmiennych wyjściowych, uzyskano najwyższy dotychczas wynik: 0.7842, co świadczyło o korzystnym wpływie dodania KNN do kompozycji modeli.

```
# Krok 7: Dodanie nowego modelu do stackingu (np. KNN)
from sklearn.neighbors import KNeighborsRegressor

targets = ["cukier", "cisnienie_skurczowe", "cisnienie_rozkurczowe"]
r2_scores_stack2 = []

for target in targets:
    stack2 = StackingRegressor(
        estimators=[
            ('rf', rf),
            ('xgb', best_xgb),
            ('knn', KNeighborsRegressor())
        ],
        final_estimator=LinearRegression()
    )

    stack2.fit(X_train, y_train[target])
    y_pred = stack2.predict(X_test)
    r2 = r2_score(y_test[target], y_pred)
    r2_scores_stack2.append(r2)

avg_r2_stack2 = np.mean(r2_scores_stack2)
results["Stacking + KNN"] = avg_r2_stack2
print(f"Stacking + KNN średni  $R^2$  score: {avg_r2_stack2:.4f}")
```

[14] ✓ 35.0s Python

... Stacking + KNN średni R^2 score: 0.7842

Rys. 7. Rozszerzono zestaw stackingowy o model KNN

8. Przetestowano modele na zbiorze klasyfikacyjnym Wine

W celu przetestowania działania stackingowego podejścia w kontekście klasyfikacji, użyto wbudowanego zbioru wine. Zastosowano modele klasyfikacyjne: RandomForestClassifier, XGBClassifier oraz StackingClassifier. Modele osiągnęły bardzo wysoką skuteczność: Random Forest i Stacking uzyskały 100% trafności, natomiast XGBoost 94.44%.

```
# Krok 8: Testowanie modeli na zbiorze danych Wine
from sklearn.datasets import load_wine

wine = load_wine()
X_wine = pd.DataFrame(wine.data, columns=wine.feature_names)
y_wine = pd.DataFrame(wine.target)

Xw_train, Xw_test, yw_train, yw_test = train_test_split(X_wine, y_wine, test_size=0.2, random_state=42)

# Ponieważ mamy klasyfikację, przetestujemy XGBoost jako klasyfikator
from xgboost import XGBClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import StackingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score

clf_rf = RandomForestClassifier(random_state=42)
clf_xgb = XGBClassifier(random_state=42)
clf_stack = StackingClassifier(
    estimators=[('rf', clf_rf), ('xgb', clf_xgb)],
    final_estimator=LogisticRegression()
)

clf_models = {
    "Random Forest (wine)": clf_rf,
    "XGBoost (wine)": clf_xgb,
    "Stacking (wine)": clf_stack
}

for name, clf in clf_models.items():
    clf.fit(Xw_train, yw_train.values.ravel())
    y_pred = clf.predict(Xw_test)
    acc = accuracy_score(yw_test, y_pred)
    results[name] = acc
    print(f"{name} Accuracy: {acc:.4f}")
```

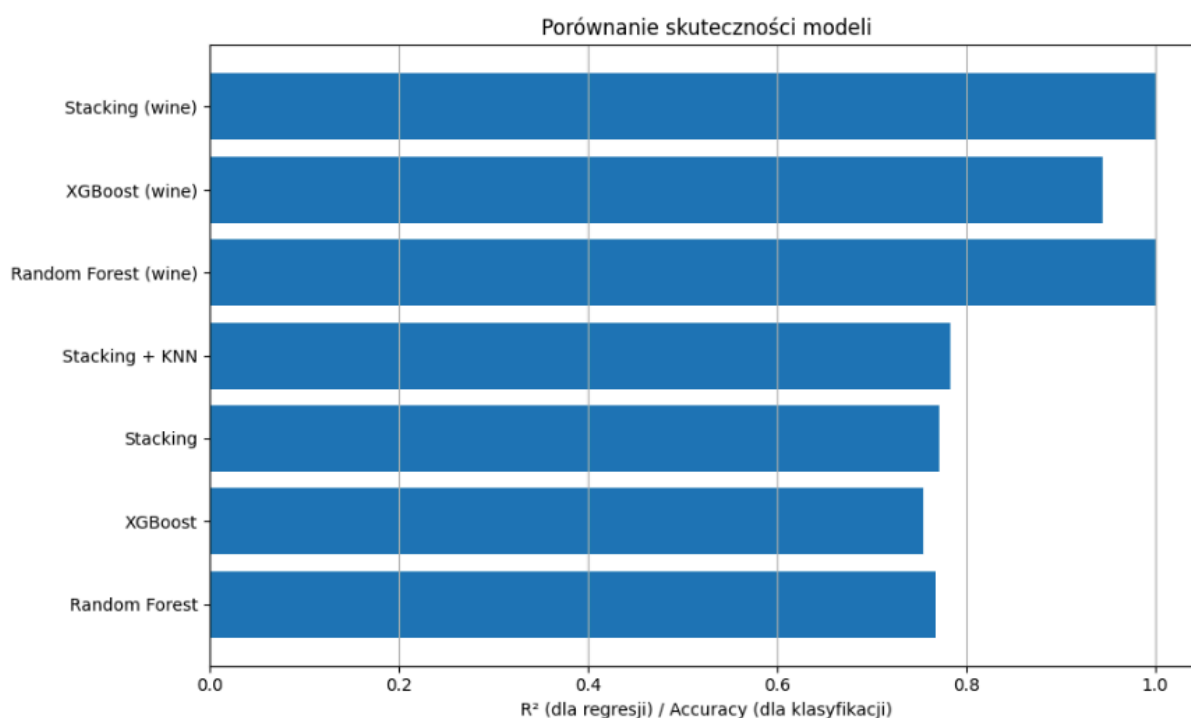
[15] ✓ 1.0s Python

```
... Random Forest (wine) Accuracy: 1.0000
... XGBoost (wine) Accuracy: 0.9444
... Stacking (wine) Accuracy: 1.0000
```

Rys. 8. Przetestowano modele na zbiorze klasyfikacyjnym Wine

9. Zwizualizowano wyniki w formie wykresu słupkowego

Wyniki wszystkich modeli (regresyjnych i klasyfikacyjnych) zaprezentowano na jednym wykresie słupkowym, porównującym średnie R^2 (dla regresji) i trafność (accuracy) dla klasyfikacji. Wykres potwierdził przewagę modeli ensemble oraz skuteczność stackingowego podejścia, szczególnie w wariancie z dodatkowym modelem KNN.



Rys. 9. Zwizualizowano wyniki w formie wykresu słupkowego

3. Wnioski

Modele ensemble, takie jak Random Forest, XGBoost i szczególnie Stacking, wykazały wysoką skuteczność w zadaniu regresji zdrowotnej, uzyskując średnie wartości współczynnika determinacji R^2 powyżej 0.75. Dodanie modelu KNN do zestawu stackingowego pozwoliło jeszcze poprawić wynik, co potwierdziło, że różnorodność estymatorów bazowych może pozytywnie wpłynąć na jakość predykcji.