Projekt Klasyfikacji Chorób Serca

Opis Projektu

Klasyfikator chorób serca oparty na zbiorze danych CDC (**319,795 próbek**). Projekt implementuje pipeline klasyfikacyjny do przewidywania występowania chorób serca na podstawie zestawu wskaźników medycznych i stylu życia.

Cel projektu

Stworzenie wydajnego modelu klasyfikacyjnego przewidującego występowanie chorób serca w populacji na podstawie danych medycznych i demograficznych.

Kluczowe Wskaźniki Wydajności (KPI)

- **F1-score (weighted) ≥ 0.75** (metryka zapisywana w MLflow)
- **LogLoss** jako wskaźnik jakości klasyfikacji probabilistycznej
- Stabilność metryk w walidacji krzyżowej (**std ≤ 0.05**)
- Automatyzacja doboru hiperparametrów przy użyciu **Optuna**
- Śledzenie zmian kodu i modeli poprzez **MLflow + Git**

Ocena Ryzyka

- **Ryzyko dysproporcji danych:** niezbalansowanie klas (91.4% „No", 8.6% „Yes") -- wymaga zastosowania balansowania (class_weight, stratified sampling)
- **Ryzyko zależności od modelu:** użycie konkretnego frameworka (CatBoost)
- **Overfitting:** kontrolowane przez early stopping i walidację krzyżową
- **Ryzyko reprodukowalności:** minimalizowane przez kontrolę wersji (joblib, MLflow, git hash, random_state)

Opis Zbioru Danych

Zbiór składa się z **319,795 próbek i 18 kolumn** (17 cech + 1 etykieta).

Kolumny:

Kolumna	Opis
HeartDisease	Diagnoza choroby serca (Yes/No) -- **zmienna docelowa**
BMI	Wskaźnik masy ciała
Smoking	Status palenia (Yes/No)
AlcoholDrinking	Spożywanie alkoholu (Yes/No)
Stroke	Przebyta udar (Yes/No)
PhysicalHealth	Dni złego zdrowia fizycznego (0--30)
MentalHealth	Dni złego zdrowia psychicznego (0--30)
DiffWalking	Trudności w chodzeniu (Yes/No)
Sex	Płeć (Male/Female)
AgeCategory	Kategoria wiekowa
Race	Rasa/pochodzenie
Diabetic	Status cukrzycy (Yes/No/Borderline/Yes during pregnancy)

PhysicalActivity	Aktywność fizyczna (Yes/No)
GenHealth	Ogólny stan zdrowia (Excellent/Very good/Good/Fair/Poor)
SleepTime	Godziny snu na dobę
Asthma	Astma (Yes/No)
KidneyDisease	Choroba nerek (Yes/No)
SkinCancer	Rak skóry (Yes/No)

Rozkład zmiennej docelowej

- No (brak choroby serca): **292,422 próbek (91.4%)**
- Yes (choroba serca): **27,373 próbek (8.6%)**

⚠ Wyraźna nierównowaga klas -- zastosowano class_weights w CatBoost i stratified sampling.

🤖 Opis Modelu

Model: **CatBoostClassifier (gradient boosting)**

Cechy CatBoost

- natywne wsparcie dla danych kategorycznych (bez one-hot encodingu)
- ordered boosting -- redukcja ryzyka overfittingu
- obsługa niezbalansowanych klas (class_weights)

🔧 Dostrajanie Hiperparametrów

- Implementacja: **Optuna** + mlflow.start_run(nested=True)
- Najlepsze parametry zapisywane w best_params.pkl i logowane w MLflow
- Kluczowe parametry optymalizacji:
  - iterations
  - learning_rate
  - depth
  - l2_leaf_reg
  - class_weights

🔁 Walidacja Krzyżowa

- Implementacja: catboost.cv z 5-fold stratified shuffle
- Monitorowane metryki: **F1, LogLoss, AUC-ROC**
- Wyniki wizualizowane z pasmami błędów przy użyciu **Plotly**

🗂️ Struktura Projektu

```
├── .devcontainer/          # konfiguracja Codespaces / Dockera
├── .github/workflows/      # pipeline CI/CD
│   ├── ci.yml              # linting i formatowanie kodu
│   └── lint-code.yml       # dodatkowe testy przy PR
├── ARISA_DSML/             # główny kod źródłowy
```

```
│   ├── __init__.py
│   ├── config.py          # konfiguracja projektu
│   ├── preproc.py         # przetwarzanie danych
│   ├── train.py           # trenowanie modeli
│   ├── predict.py         # predykcje i monitoring
│   ├── resolve.py         # zarządzanie modelami
│   └── helpers.py         # funkcje pomocnicze
├── data/                  # dane projektu
│   ├── raw/               # dane surowe z Kaggle
│   ├── interim/           # dane pośrednie
│   ├── processed/         # dane przetworzone
│   └── external/          # dane zewnętrzne
├── models/                # modele i artefakty
├── reports/               # raporty i wizualizacje
│   └── figures/           # wykresy i diagramy
├── results/               # wyniki eksperymentów EDA/DS
├── mlruns/                # MLflow tracking
├── mlartifacts/           # MLflow artifacts
├── notebooks/             # notatniki Jupyter
│   ├── 01-Before_MLOps.ipynb
│   └── 02-Model_version.ipynb
├── tests/                 # testy jednostkowe
├── docs/                  # dokumentacja
├── Makefile               # automatyzacja zadań
├── README.md              # opis projektu
├── pyproject.toml         # konfiguracja pakietu
├── setup.cfg              # konfiguracja narzędzi
├── requirements.txt       # zależności Python
```

└── .gitignore # wykluczenia z git

☸ Wymagania wstępne

- Python 3.11+
- Pandas & NumPy
- Scikit-learn
- Matplotlib & Plotly
- Jupyter Notebook
- MLflow
- Git & GitHub

🚀 Instalacja i Uruchomienie

Klonowanie repozytorium

```
git clone https://github.com/Pawel20240101/PZ_ARISA_MLOps_Final.git

cd PZ_ARISA_MLOps_Final
```

Tworzenie środowiska

```
python -m venv .venv

# Windows

.\.venv\Scripts\activate

# Linux/Mac

source .venv/bin/activate
```

Instalacja zależności

```
pip install -r requirements.txt
```

Dane wejściowe

Dane są automatycznie pobierane z Kaggle przez API w preproc.py

Start MLflow UI

```
mlflow ui
```

Lub

mlflow ui --host 127.0.0.1 --port 5000

🔄 Pipeline MLOps

Makefile - Lokalne testy i rozwój

Pełny pipeline lokalny (rozwój i debugowanie)

make test # Kompletny test: linting + pipeline ML

Poszczególne etapy

make preprocess # Pobieranie i przetwarzanie danych

make train # Trenowanie modelu z hyperopt

make predict # Predykcje i monitoring drift

Jakość kodu

make lint # Sprawdzenie formatowania

make format # Automatyczne formatowanie

Uwaga: make test uruchamia pełny pipeline ML (preprocess → train → predict) i jest używany lokalnie w CodeSpace do rozwoju i debugowania.

GitHub Actions - CI/CD

ci.yml (uruchamia się przy push/PR na main i test)

- Sprawdzenie formatowania: black --check

- Linting: flake8

- Sortowanie importów: isort --check-only

- Testy jednostkowe: pytest -v

lint-code.yml (uruchamia się przy PR na main)

- Instalacja zależności: make requirements

- Linting: make lint

Kluczowa różnica:

- **GitHub Actions** = tylko jakość kodu (szybkie, bez danych ML)
- **Makefile** = pełny pipeline ML (lokalne środowisko)

🖌️ Przetwarzanie Danych

- Konwersja targetu (Yes/No → 1/0)
- Podział train/test (stratyfikowany)
- Walidacja kolumn kategorycznych
- Balansowanie klas: **class_weight + stratified sampling**

📈 Metryki Ewaluacji

Monitorowane:

- **F1-score (weighted)**
- Precision i Recall dla klasy pozytywnej
- AUC-ROC
- Confusion Matrix
- LogLoss

📡 Monitorowanie i Wsparcie

- **MLflow** -- śledzenie metryk i wersji modeli
- **NannyML** -- wykrywanie driftu danych
- **Git** -- kontrola wersji

🩹 Jakość Kodu

- **Formatowanie:** black (line-length 99)
- **Linting:** flake8
- **Sortowanie importów:** isort
- **Testy jednostkowe:** pytest (w katalogu tests/)
- **Dokumentacja:** docstringi + README.md

🔒 Git Management

.gitignore - Wykluczenia

Duże pliki ML (nie synchronizowane)

data/ # Dane pobierane przez API Kaggle

models/ # Modele generowane podczas treningu

mlruns/ # MLflow tracking data

mlartifacts/ # MLflow artifacts

reports/ # Raporty i wykresy

Pliki systemowe

__pycache__/

*.pyc

.venv/

.pytest_cache/

Korzyści:

- Repozytorium lekkie (kilka MB zamiast setek MB)
- Szybkie klonowanie i CI/CD
- Dane odtwarzalne przez make preprocess

📊 Wyniki Eksperymentów

Cross-Validation (N=5)

- Mean F1 Score: (wymaga poprawy - niezbalansowane klasy)
- Mean LogLoss: ~0.49 (po zbieżności)
- Standard deviation ≪ 0.05 (brak oznak overfittingu)

Analiza SHAP

- Najważniejsze cechy: AgeCategory, GenHealth, Stroke, BMI
- Cechy o małym wpływie: Race → model nie dyskryminuje

📜 Wnioski i Rekomendacje

Obserwacje:

- F1-score poniżej docelowego KPI (≥0.75)
- Silne niezbalansowanie klas wymaga dodatkowych technik

Rekomendacje poprawy:

- Zastosowanie SMOTE/ADASYN do generowania syntetycznych próbek
- Optymalizacja threshold klasyfikacji
- Ensemble methods (voting, stacking)
- Feature engineering (interakcje między cechami)

Wnioski medyczne:

- Najważniejsze czynniki ryzyka: starszy wiek, słaby stan zdrowia, historia udaru, wysokie BMI
- Interpretowalność zapewniona dzięki SHAP
- Model wymaga dalszej optymalizacji dla zastosowań klinicznych