Mini-AutoML: Raport z implementacji systemu

Mikołaj Twardowski, Maciej Turczyński, Mateusz Kiszka

25 stycznia 2026

1 Wstęp

Niniejszy raport dokumentuje implementację systemu `Mini-AutoML`, zaprojektowanego do automatyzacji procesu klasyfikacji binarnej danych tabelarycznych. Zamiast klasycznego przeszukiwania przestrzeni hiperparametrów, przyjęto podejście wykorzystujące predefiniowane portfolio modeli wygenerowane na bazie benchmarków MementoML [2]. Kluczowym elementem architektury jest dwuetapowy mechanizm selekcji: szybki ranking kandydatów z użyciem strategii *budget-aware pruning* oraz finalna agregacja predykcji w formie ważonego *ensemble*. Takie rozwiązanie pozwala na osiągnięcie wysokiej zdolności generalizacji przy minimalnym nakładzie obliczeniowym.

2 Etap 1: Konstrukcja Portfolio Modeli

Jak pisaliśmy we wstępie, zamiast manualnego, arbitralnego doboru parametrów, portfolio zostało wygenerowane w oparciu o dane z projektu *MementoML*, udostępniającego rezultaty wydajności tysięcy konfiguracji algorytmów na zróżnicowanych zbiorach danych OpenML [2] oraz z [4], AMLB benchmarku . Proces konstrukcji portfolio został zaimplementowany w środowisku Python i składał się z trzech kluczowych etapów: normalizacji, selekcji oraz translacji parametrów.

2.1 Przetwarzanie Danych i Normalizacja Wyników

Na początku przetworzyliśmy pliki źródłowe zawierające łącznie ponad 7000 unikalnych konfiguracji hiperparametrów dla rodzin: CatBoost, XGBoost, GBM, Random Forest, Ranger, GLMnet i kilka więcej.

Surowe wyniki metryki AUC z bazy MementoML nie są bezpośrednio porównywalne między różnymi zbiorami danych – łatwe problemy klasyfikacyjne naturalnie generują wysokie wyniki, co mogłoby faworyzować modele testowane na takich zbiorach. Aby wyeliminować to obciążenie i zidentyfikować konfiguracje rzeczywiście uniwersalne, zaimplementowaliśmy metodę **Relative Performance Score**.

Dla każdego modelu m na danym podziale zbioru danych d (split), obliczyliśmy znormalizowany wynik względem najlepszego rezultatu w danym eksperymencie:

$$Score_{norm}(m, d) = \frac{AUC(m, d)}{\max_{m' \in M_d} AUC(m', d)}, \tag{1}$$

gdzie M_d to zbiór wszystkich modeli testowanych na danym podziale d. Wynik 1.0 oznacza, że dana konfiguracja była bezkonkurencyjna w danej próbie. Ostatecznym kryterium rankingu była średnia z tak znormalizowanych wyników, co pozwoliło wyłonić modele stabilne na różnorodnych charakterystykach danych.

2.2 Translacja R → Python

Baza MementoML opiera się na ekosystemie języka R (pakiet `mlr`). Kluczowym elementem tej części projektu było stworzenie dedykowanego parsera mapującego parametry R na bibliotekę `scikit-learn`, `xgboost` i `catboost`. Proces ten wymagał nie tylko mapowania nazw, ale także transformacji wartości i typów danych.

Zaimplementowaliśmy następujące reguły translacji:

- **GLMnet → LogisticRegression:** Kluczowym wyzwaniem była inwersja parametru regularyzacji. Parametr `lambda` z R został przeliczony na parametr C w `sklearn` wg wzoru:

$$C = \begin{cases} \frac{1}{\lambda} & \text{dla } \lambda > 0 \\ 1.0 & \text{dla } \lambda = 0 \end{cases}$$

  Dodatkowo, parametr `alpha` (mixing percentage) został zmapowany na odpowiednie typy kar: `l1` (dla $\alpha = 1$), `l2` (dla $\alpha = 0$) oraz `elasticnet`.

- **Ranger → RandomForestClassifier:** Zmapowano kluczowe parametry wpływające na strukturę drzewa:
  - `num.trees` → `n_estimators`.
  - `mtry` → `max_features`. Parser obsługuje tu zarówno wartości liczbowe (int), jak i oznaczenia symboliczne (np. transformacja stringa z R na odpowiednik w Pythonie).
  - `min.node.size` → `min_samples_leaf`.
  - `replace` → `bootstrap` (mapowanie logiczne TRUE/FALSE).

- **XGBoost i Boosting:** Dokonano mapowania parametrów sterujących procesem uczenia, m.in.: `nrounds` → `n_estimators`, `eta` → `learning_rate` oraz `interaction.depth` → `max_depth`. Dodano również stałe parametry techniczne (np. `eval_metric='logloss'`, `use_label_encoder=False`) w celu zapewnienia kompatybilności z najnowszymi wersjami bibliotek.

Dodatkowo zadbaliśmy o poprawność techniczną zapisu plików. Ponieważ standardowy format JSON nie obsługuje specyficznych formatów liczbowych używanych przez biblioteki obliczeniowe (takie jak NumPy), napisaliśmy prostą funkcję pomocniczą. Automatycznie zamienia ona te wartości na zwykłe liczby zrozumiałe dla Pythona, co zapobiega błędom podczas generowania pliku konfiguracyjnego.

3 Etap 3: Implementacja Systemu Mini-AutoML

3.1 Preprocessing danych

Preprocessing jest bardzo minimalistyczny. Dla numerycznych kolumn jest to tylko SimpleImputer na medianie i RobustScaler. Wiele różnorodnych kombinacji było testowanych, od quantileTransformera w dane normalne, SelectKBest przy funkcji mutual-info, używając polynomialfeatures by wynajdywać więcej zmiennych, po klasyczne PCA. Jednakże, nieważne czy preprocessing miał 10 kroków, czy tylko 3 , z jakiegoś powodu wyniki na przykładowym zbiorze danych zawsze były niższe, dlatego postanowiliśmy na prosty preprocessing. Tak, ten 'pakiet', miał być do dowolnego zbioru danych, nie do przykładowego trudnego zbioru danych z githuba, jednakże doszliśmy do wniosku że te niższe wyniki coś muszą oznaczać i może warto się posłuchać trudnego datasetu niż prostszych.

Główna klasa `MiniAutoML` realizuje proces uczenia w dwóch fazach, z naciskiem na efektywność czasową.

3.2 Smart Ranking (Pruning)

Podczas wstępnej selekcji (metoda `fit`) system stosuje strategię *Budget-Aware Pruning*. Modele są uruchamiane na zmniejszonej liczbie iteracji (np. `n_estimators=200` dla modeli boostingowych), co pozwala na szybką estymację ich potencjału bez pełnego obciążenia obliczeniowego. Zastosowaliśmy do tego 3-krotną walidację krzyżową (StratifiedKFold).

3.3 Ensembling (Soft Voting)

W fazie finałowej wybieranych jest 5 najlepszych, unikalnych rodzin modeli. Ich predykcje są łączone metodą ważonego głosowania (Weighted Soft Voting), gdzie wagi zależą wykładniczo od wyniku w rankingu: $W_i = \exp(Score_i \cdot 10)$. Takie podejście promuje modele wybitne, ale pozwala słabszym korygować błędy w trudnych przypadkach.

Doświadczenie z pierwszego projektu [3] i literatury [1] wskazuje, że różne rodziny algorytmów posiadają odmienną charakterystykę błędów oraz „tunowalność". Aby zapewnić różnorodność niezbędną dla efektywnego ensemblingu w późniejszym etapie, zamiast wybierać globalnie najlepsze 50 modeli (co mogłoby poskutkować dominacją np. samej rodziny CatBoost), ostawiono na różnorodność rodzin w finalnym ensemblingu.

4 Wnioski i Podsumowanie

Realizacja projektu `Mini-AutoML` pozwoliła na wyciągnięcie następujących wniosków:

- **Wyzwania techniczne (R vs Python):** Wymagającym elementem projektu nie była tylko teoria, ale i inżynieria danych. Konieczne było stworzenie niezawodnego tłumacza parametrów z języka R (biblioteka `mlr`) na Python (`scikit-learn`). Kluczowe dla stabilności systemu okazało się też automatyczne naprawianie formatów liczbowych (konwersja typów `numpy` do JSON), co zapobiega błędom zapisu.
- **Szybkość i różnorodność:** Dzięki technice „przycinania" (testowanie modeli na mniejszej liczbie drzew lub iteracji) system mógł sprawdzić wiele konfiguracji w bardzo krótkim czasie. Z kolei nałożenie limitów (kwot) na poszczególne rodziny modeli wymusiło różnorodność. Dzięki temu końcowy ensemble składa się z odmiennych algorytmów, co zazwyczaj daje lepsze wyniki niż łączenie modeli bardzo do siebie podobnych.
- **Możliwości rozwoju:** Obecnie system korzysta ze stałej, uniwersalnej listy 50 modeli. W przyszłości można by to ulepszyć, dobierając początkową pulę modeli dynamicznie – w zależności od cech konkretnego zbioru danych .

Osiągnięty wynik **Balanced Accuracy na poziomie 0.57** (na podesłanym zbiorze danych) potwierdza, że cały pipeline działa poprawnie technicznie. Wynik ten sugeruje również, że istnieje jeszcze pole do poprawy i dalszego rozwoju systemu.

Literatura

[1] P. Probst, B. Bischl, A-L. Boulesteix, (2019), *Tunability: Importance of Hyperparameters of Machine Learning Algorithms*, `https://jmlr.org/papers/volume20/18-444/18-444.pdf`

[2] W. Kretowicz, P. Biecek, (2020), *MementoML: Performance of selected machine learning algorithm configurations on OpenML100 datasets*, `https://arxiv.org/abs/2008.13162`, `https://www.kaggle.com/datasets/mi2datalab/mementoml`

[3] M. Twardowski, M. Turczyński, M. Kiszka, (2025), *Badanie tunowalności hiperparametrów*, Automatyczne Uczenie Maszynowe, Politechnika Warszawska

[4] P. Gijsbers, M. L. P. Bueno, (2022) *AMLB: an AutoML Benchmark*, `https://arxiv.org/abs/2207.12560`