

Mini-AutoML dla Danych Tabelarycznych

Julia Dudzińska, Weronika Gozder, Mikołaj Ozimek

26 stycznia 2026

Spis treści

1	Wstęp	1
2	Selekcja modeli	1
2.1	Benchmark MementoML	1
2.2	Proces selekcji konfiguracji	1
2.2.1	Miara wykorzystana do selekcji modeli	2
2.2.2	Wynikowe portfolio modeli	2
3	Implementacja systemu MiniAutoML	2
3.1	Preprocessing danych	3
3.2	Strategie selekcji modeli	3
3.3	Metody ensemblingu	4
3.4	Eksperymenty	4
4	Podsumowanie	4

1 Wstęp

Celem projektu jest implementacja uproszczonego systemu automatycznego uczenia maszynowego MiniAutoML do binarnej klasyfikacji na danych tabelarycznych. Nasza implementacja wykorzystuje portfolio modeli stworzone poprzez screening na podstawie wyników zewnętrznych (MementoML) oraz umożliwia 3 sposoby selekcji modeli (CV, stabilność, heurystyczna) i 3 typy ensemblingu (soft voting, hard voting, stacking).

2 Selekcja modeli

Do budowy portfolio modeli zastosowano podejście oparte na wykorzystaniu wyników z zewnętrznych benchmarków (Opcja B), co pozwoliło uniknąć czasochłonnych eksperymentów lokalnych. Jako źródło danych wykorzystano benchmark MementoML [4], który zawiera kolekcję wyników testów siedmiu popularnych algorytmów uczenia maszynowego na 39 zbiorach danych z platformy OpenML [5], a konkretnie ze zbioru benchmarkowego OpenML100 zawierającego zbiory danych do klasyfikacji binarnej. MementoML obejmuje łącznie 3 787 569 ewaluacji modeli.

2.1 Benchmark MementoML

Benchmark MementoML [4] obejmuje algorytmy `catboost`, `gbm`, `xgboost`, `glmnet`, `kknn`, `randomForest` oraz `ranger`. W procesie tworzenia, każdy zbiór danych został podzielony na 20 par treningowo-testowych, a zestawy hiperparametrów dla każdego modelu zostały wylosowane z zdefiniowanych zakresów przed rozpoczęciem obliczeń. Dla każdej konfiguracji modelu zmierzono wydajność (ACC i AUC) oraz czas uczenia.

2.2 Proces selekcji konfiguracji

Wykorzystanie benchmarku MementoML wymagało mapowania funkcji oraz hiperparametrów z formatów używanych w pakietach R do odpowiedników w bibliotece scikit-learn (np. dla lasu losowego: `ranger` → `RandomForest`, `num.trees/ntree` → `n_estimators`). Część modeli została pominięta z dalszej analizy ze względu na brak odpowiedników w Pythonie (np. `agtboost`), problemy z instalacją (`catboost`) albo brakujące konfiguracje hiperparametrów w zbiorze danych benchmarku (`svm`). Dodatkowo, by zapewnić różnorodność modeli, zostało nałożone ograniczenie na maksymalnie 10 modeli danego typu (dla lasów losowych, `ranger` i `randomForest` zostały potraktowane jako jeden typ), z wyjątkami wynikającymi z analizy wyników (maksimum dla `XGBoost` zostało zwiększone do 15 ze względu na to że osiągał najlepsze wyniki, maksimum dla KNN zostało zmniejszone do 5 jako że najlepsze konfiguracje zwykle się duplikowały, jako że zestawy hiperparametrów w benchmarku okazały się nie być unikatowe). Modele zostały ustawione w ranking na podstawie agregacji wyników dla każdej kombinacji (model, konfiguracja hiperparametrów). Dla każdej konfiguracji obliczono AUC_{mean} (średnie AUC po wszystkich zbiorach danych i podziałach treningowo-testowych), AUC_{std} (odchylenie standardowe AUC) $CV = \frac{AUC_{std}}{AUC_{mean}}$ (współczynnik zmienności).

2.2.1 Miara wykorzystana do selekcji modeli

Zamiast selekcji wyłącznie na podstawie średniej wartości AUC, zastosowano miarę kompozytową uwzględniającą wydajność i stabilność modeli w proporcji 70-30. Miara ta została zdefiniowana jako:

$$\text{Score}_{\text{comp}} = 0.7 \cdot \text{AUC}_{\text{norm}} + 0.3 \cdot (1 - \text{CV}_{\text{norm}}) \quad (1)$$

gdzie:

- $\text{AUC}_{\text{norm}} = \frac{\text{AUC}_{\text{mean}} - \text{AUC}_{\text{min}}}{\text{AUC}_{\text{max}} - \text{AUC}_{\text{min}}}$ – znormalizowana średnia AUC
- $\text{CV}_{\text{norm}} = \frac{\text{CV} - \text{CV}_{\text{min}}}{\text{CV}_{\text{max}} - \text{CV}_{\text{min}}}$ – znormalizowany współczynnik zmienności

2.2.2 Wynikowe portfolio modeli

Finalnie wyselekcjonowane portfolio zawiera 50 konfiguracji modeli, których zagregowane dane przedstawione zostały w Tabeli 1.

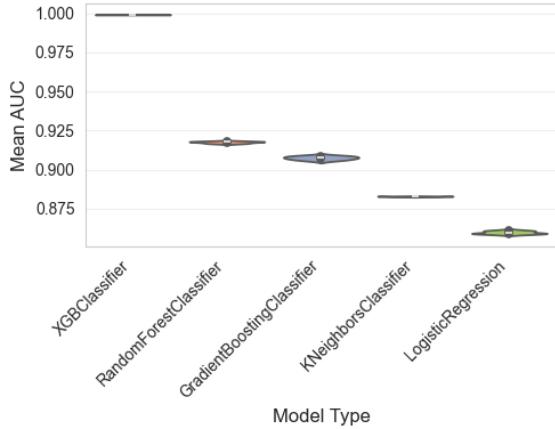
Tabela 1: Rozkład modeli w finalnym portfolio

Typ modelu	Liczba	AUC _{mean}	CV _{mean}	Score _{comp}
XGBClassifier	15	0.9992	0.0003	0.9997
GradientBoostingClassifier	10	0.9075	0.1147	0.7811
LogisticRegression	10	0.8597	0.1556	0.6821
RandomForestClassifier	10	0.9175	0.1079	0.8006
KNeighborsClassifier	5	0.8828	0.1351	0.7305

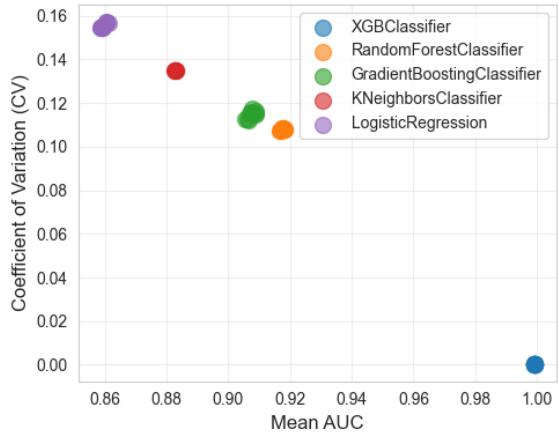
Konfiguracje algorytmu XGBoost wypadły zdecydowanie najlepiej, osiągając najwyższe wartości zarówno średniego AUC (≈ 0.999), jak i najniższy współczynnik zmienności (≈ 0.0003 , wartość o 3 rzędy mniejsza niż dla pozostałych modeli), co świadczy o ich dużej stabilności i wydajności na różnych zbiorach danych – pokazuje to zarówno Tabela 1 jak i Rysunek 1 (stanowiący ilustrację do tabeli). Najmniej różnorodne są konfiguracje KNN, które w wynikowym portfolio wszystkie przyjęły te same hiperparametry poza liczbą sąsiadów (co zostało wymuszone przez naszą implementację niedopuszczającą tych samych zestawów hiperparametrów) – ten parametr przyjął jednak kolejne wartości z zakresu 26 do 30. Rysunek 2 ilustruje trade-off między wydajnością (AUC) a stabilnością (CV) wybranych modeli. Modele XGBoost dominują w prawym dolnym rogu, podczas gdy LogisticRegression wykazuje największą zmienność.

3 Implementacja systemu MiniAutoML

System MiniAutoML został zaprojektowany jako uproszczony framework AutoML do klasyfikacji binarnej na danych tabelarycznych. Główne funkcjonalności obejmują automatyczne przetwarzanie danych, 3 metody selekcji modeli z portfolio oraz 3 typyensemblingu.



Rysunek 1: Wykres skrzypcowy przedstawiający rozkład AUC dla modeli w portfolio



Rysunek 2: Wykres punktowy przedstawiający AUC_{mean} vs CV dla modeli w portfolio

3.1 Preprocessing danych

Automatyczne przetwarzanie danych wykorzystuje mechanizm wieloetapowych transformacji, które są konstruowane dynamicznie na podstawie wykrytych typów zmiennych w zbiorze treningowym. Dla zmiennych numerycznych stosowana jest imputacja braków wartością medianą oraz standaryzacja. Zmienne kategoryczne poddawane są imputacji najczęściej występującą wartością, a następnie kodowane za pomocą one-hot encoding.

3.2 Strategie selekcji modeli

System implementuje trzy strategie selekcji modeli, które można wybrać za pomocą parametru konfiguracyjnego. Pierwsza strategia (`cross_val`), oparta na walidacji krzyżowej, przeprowadza 5-krotną CV dla każdego modelu z portfolio i rankinguje je według średniej wartości metryki *balanced accuracy*. Druga (`stability`) koncentruje się na stabilności modeli, wykorzystując nie tylko średnią wydajność, ale również jej wariancję między foldami walidacji krzyżowej. Dla każdego modelu obliczana jest agregowana miara jakości, która nagradza wysoką średnią wydajność oraz penalizuje wysokie odchylenie standardowe. Trzecia strategia (`heuristic`) wykorzystuje heurystyki oparte na charakterystykach zbioru danych, takich jak liczba próbek czy wymiarowość przestrzeni. Na podstawie heurystycznych reguł (np. modele z regularizacją są lepsze dla wielowymiarowych zbiorów danych) modele otrzymują wstępne oceny, następnie 10 najlepszych kandydatów jest walidowanych za pomocą CV. Wszystkie strategie stosowane są dwuetapowo w celu optymalnego wykorzystania zasobów i przeanalizowania możliwie dużej liczby konfiguracji modeli w jak najkrótszym czasie. W pierwszym etapie metoda wywoływana jest jedynie na podzbiorze danych wejściowych o ustalonym rozmiarze (domyślnie 50%), jeśli to możliwe podzbior ten wyłaniany jest z zachowaniem proporcji klas. Jeżeli w zadany czasie trwania całego zadania uda się przeprowadzić pierwszy etap dla wszystkich konfiguracji, metoda wywoływana jest ponownie dla N najlepszych konfiguracji, tym razem na całym zbiorze danych. N jest równe podwojonej liczbie metod oczekiwanych w ensemblingu (czyli $2 \leq N \leq 10$). Pozycja tych konfiguracji jest aktualizowana w rankingu na podstawie ich skuteczności w drugim etapie.

3.3 Metody ensemblingu

Finalny predyktor może być zarówno pojedynczym najlepszym modelem, jak i zespołem do pięciu top modeli z rankingu. System oferuje trzy strategie konstrukcji ensemble: soft voting, hard voting i stacking. Metoda soft voting agreguje prawdopodobieństwa przewidywane przez poszczególne modele, obliczając ważone średnie dla każdej klasy, hard voting zlicza głosy poszczególnych modeli na klasy, a stacking wykorzystuje dwupoziomową architekturę, gdzie predykcje z modeli bazowych stają się cechami dla meta-learnera.

3.4 Eksperymenty

W celu ustalenia domyślnej konfiguracji systemu **MiniAutoML** przeprowadzono eksperymenty z użyciem biblioteki **Optuna** [2]. W tym celu przeprowadzono 20 prób eksperymentalnych optymalizujących *balanced accuracy* na zbiorze **diabetes** z OpenML [1, 3]. Optymalny wynik metryki, 0.7425, powtórzył się dla więcej niż jednej próby. Z racji, że kluczowym zadaniem projektu była selekcja modeli, zdecydowaliśmy się wybrać konfigurację ze strategią selekcji, która powtarzała się wśród tych prób najczęściej. Jedna z wyłonionych konfiguracji stosowała również ensembling. Jako, że można się spodziewać, że łączenie predykcji ogólnie rzecz biorąc wpływa korzystnie na skuteczność, wspomniana konfiguracja została ustanowiona domyślną. Tabela 2 przedstawia badane parametry wraz z ich zakresami oraz wyznaczoną optymalną (domyślną) wartością.

Tabela 2: Parametry systemu **MiniAutoML** wraz z ich zakresem badanym w eksperymentach z użyciem biblioteki **Optuna** oraz wyznaczoną optymalną wartością.

Parametr	Badany zakres	Optymalna wartość
Strategia selekcji	[‘cross_val’, ‘stability’, ‘heuristic’]	stability
Liczba foldów walidacji krzyżowej	3 - 5	3
Liczba konfiguracji do ensemblingu	1 - 5	2
Strategia ensemblingu	[‘soft_voting’, ‘hard_voting’, ‘stacking’]	‘hard_voting’
Próg stabilności dla strategii stability	0.0 - 1.0	0.281

4 Podsumowanie

W ramach projektu zaimplementowano system **MiniAutoML** do binarnej klasyfikacji tabelarycznej, wykorzystujący portfolio 50 top modeli wyselekcjonowanych z benchmarku MementoML za pomocą kompozytowej miary wydajności i stabilności (70-30 AUC/CV). System oferuje trzy strategie selekcji (CV, stabilność, heurystyczna) oraz ensembling (soft/hard voting, stacking), z domyślną konfiguracją zoptymalizowaną eksperymentalnie na zbiorze **diabetes**. Podejście to zapewnia efektywność bez kosztownych benchmarków lokalnych, osiągając wysoką stabilność i użyteczność w warunkach ograniczonych zasobów.

Literatura

- [1] OpenML diabetes. https://www.openml.org/search?type=data&sort=nr_of_likes&status=active&id=37. Accessed: 2026-01-26.
- [2] Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.
- [3] Matthias Feurer, Jan N. van Rijn, Arlind Kadra, Pieter Gijsbers, Neeratyoy Mallik, Sahithya Ravi, Andreas MĂžller, Joaquin Vanschoren, and Frank Hutter. Openml-python: an extensible python api for openml. *Journal of Machine Learning Research*, 22(100):1–5, 2021.
- [4] Wojciech Kretowicz and Przemysław Biecek. Mementoml: Performance of selected machine learning algorithm configurations on openml100 datasets, 2020.
- [5] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *SIGKDD Explor. Newsl.*, 15(2):49–60, June 2014.