AutoML Projekt 2

Maksim Razantsau, Oleksii Vinichenko, Yahor Lahunovich

26 stycznia 2026

1 Wstęp

Celem projektu jest napisanie prostej biblioteki automatycznego uczenia maszynowego dla rozwiązania problemu klasyfikacji binarnej na danych tabelarycznych.

2 Wybór modeli (Opcja B)

Jako modele przykładowe wybraliśmy modele zaproponowane na githubie: LogisticRegression, RandomForestClassifier oraz LGBMClassifier

Żeby zapewnić dobre działanie biblioteki na zróżnicowanych zbiorach danych przeanalizowaliśmy wyniki z ogólnodostępnych źródeł (Benchmarki OpenML, TabArena, MementoML) aby wybrać modele dla naszego systemu. Poniższa tabela przedstawia konfiguracje modeli wykorzystane w eksperymentach. Dla każdego modelu wskazano jego pochodzenie oraz kluczowe hiperparametry.

Tabela 1: Szczegółowa specyfikacja modeli.

#	Nazwa modelu	Pochodzenie	Algorytm	Kluczowe parametry
0	KKNN_1952_CLASS	TabArena	kNN	$k = 30, p = 2, weights = distance$
1	catboost_tabarena_bal	TabArena	CatBoost	$lr = 0.03, depth = 6, l2 = 3.0$
2	catboost_tabarena_deep	TabArena	CatBoost	$lr = 0.01, depth = 8, grow = Depthwise$
3	catboost_tabarena_fast	TabArena	CatBoost	$lr = 0.1, depth = 4, l2 = 5.0$
4	xt_tabarena_default	TabArena	ExtraTrees	$n\_est = 50, feat = sqrt, split = 2$
5	xt_tabarena_heavy	TabArena	ExtraTrees	$n\_est = 50, feat = 1.0, split = 16$
6	xt_tabarena_consv	TabArena	ExtraTrees	$n\_est = 100, feat = 0.5, split = 32$
7	RF_tabarena_balanced	TabArena	RF	$n\_est = 200, crit = gini, feat = sqrt$
8	RF_tabarena_Entropy	TabArena	RF	$n\_est = 300, crit = entropy, cw = bal\_sub$
9	KNN_Manhattan_Uniform	TabArena	kNN	$k = 15, p = 1, weights = uniform$
10	KNN_Local_Distance	TabArena	kNN	$k = 5, p = 2, weights = distance$
11	NaiveBayes_Gaussian	TabArena	NB	$var\_smoothing = 1e-9$
12	RANGER_1952_CLASS	MementoML	ExtraTrees	$n\_est = 674, crit = entropy, leaf = 1$
13	RANGER_1954_CLASS	MementoML	ExtraTrees	$n\_est = 3008, crit = entropy, leaf = 3$
14	RANGER_1978_CLASS	MementoML	ExtraTrees	$n\_est = 568, boot = True, leaf = 1$
15	GLMNET_1171_CLASS	MementoML	LogReg	$elasticnet, C = 376, l1 = 0.73$
16	GLMNET_1337_CLASS	MementoML	LogReg	$elasticnet, C = 414, l1 = 0.53$
17	GLMNET_1960_CLASS	MementoML	LogReg	$elasticnet, C = 320, l1 = 0.68$
18	GBM_1101_CLASS	MementoML	GBM	$n\_est = 3192, lr = 0.01, depth = 5$
19	GBM_1272_CLASS	MementoML	GBM	$n\_est = 8039, lr = 0.005, depth = 5$
20	GBM_1399_CLASS	MementoML	GBM	$n\_est = 9751, lr = 0.003, depth = 5$

Tabela 1 - ciąg dalszy

	Nazwa modelu	Pochodzenie	Algorytm	Kluczowe parametry
21	LGBM_TD_CLASS_def	OpenML	LightGBM	$n\_est = 1000, lr = 0.04, leaves = 50$
22	LGBM_TD_REG	OpenML	LightGBM	$n\_est = 1000, lr = 0.05, leaves = 100$
23	LGBM_GOSS	OpenML	LightGBM	$boost = goss, lr = 0.05, depth = 8$
24	LGBM_ExtraTrees	OpenML	LightGBM	$extra = True, lr = 0.03, leaves = 64$
25	XGB_TD_CLASS	OpenML	XGBoost	$n\_est = 1000, lr = 0.08, depth = 6$
26	XGB_TD_REG	OpenML	XGBoost	$n\_est = 1000, lr = 0.05, depth = 9$
27	XGB_PBB_CLASS	OpenML	XGBoost	$n\_est = 4168, lr = 0.018, depth = 13$
28	CB_TD_CLASS	OpenML	CatBoost	$n\_est = 1000, lr = 0.08, depth = 7$
29	CB_TD_REG	OpenML	CatBoost	$n\_est = 1000, lr = 0.09, depth = 9$
30	cnae-9_best_auc_MLP	OpenML	MLP	$layers = [100, 100, 100], alpha = 0.001$
31	wilt_best_auc_MLP	OpenML	MLP	$layers = [100], alpha = 0.0001$
32	credit_approval_best	OpenML	HistGB	$lr = 0.1, l2 = 100.0, max\_nodes = 128$
33	churn_best_auc	OpenML	HistGB	$lr = 0.1, max\_bins = 32, depth = 10$

2.1 Zbiory danych dla Eksperymentów

Przez nas również wybrano zbiory danych dla dalszych esperymentów systemu oraz porównania naszych modeli z modelami przykładowymi.

- **Titanic:** 892 rows.
- **Bank Marketing:** 45212 rows.
- **Spaceship Titanic:** 8694 rows
- **Adults:** 48843 rows

W większości eksperymentów wykorzystaliśmy zbiór danych titanic, ponieważ zawiera on najmniejszą liczbę obserwacji, natomiast pozostałe zbiory danych zostały użyte do końcowego uruchomienia systemu

3 Architektura Systemu

3.1 Pipeline

Do trenowania modeli z portfolio potrzebujemy pipeline'u, żeby zautomatyzować ten proces. Potrzebujemy również żeby pipeline działał na ilościowych oraz kategorycznych zmiennych.

- na ilościowych zmiennych używamy SimpleImputer, do zamiany brakujących wartości na medianę oraz RobustScaler do standartyzacji.
- na kategoryczych zmiennych używamy SimpleImputer oraz TargetEncoder do kodowania cech nominalnych.

3.2 Metoda Selekcji Modeli

Przygotowaliśmy plik models.json do pracy na CPU oraz models_gpu.json do pracy na GPU, tak aby możliwe było łatwe przełączenie pliku i uruchomienie całego systemu na GPU.

Selekcja modeli realizowana jest w metodzie fit(). Sama metoda fit() składa się z następujących kroków:

Krok 1: Trenowanie Wszystkich Modeli

Dla każdego modelu z portfolio wykonywana jest 5-krotna walidacja krzyżowa(jako prosty meta-learner) z metryką balanced_accuracy.

Aby przyspieszyć trenowanie wszystkich modeli, użyliśmy programowania asynchronicznego, dzięki czemu modele trenowały się równolegle. Znacząco przyspieszyło to cały proces trenowania

Krok 2: Ensemble Modeli

Modele są sortowane według średniego wyniku balanced_accuracy z walidacji krzyżowej. Z rankingu wybierane są 5 najlepszych modeli do dalszego ensemblingu. Rozwiązanie końcowe to stacking pięciu najlepszych modeli z regresją logistyczną jako final estimator.

3.3 Eksperymenty

Przeprowadziliśmy eksperymenty na różnych zbiorach danych w celu ulepszenia accuracy.

Eksperyment 1: RandomForest jako Feature Selector
Zbadano wpływ zastosowania wstępnej selekcji cech (Feature Selection) opartej na ważności cech z modelu RandomForest przed właściwym trenowaniem portfolio.

- Bez selekcji cech: Balanced Accuracy = 0.832
- Z selekcją cech przez RandomForest: Balanced Accuracy = 0.828
- **Wnioski:** Agresywna selekcja cech na tym etapie pogorszyła wyniki końcowe. Sugeruje to, że modele w portfolio (szczególnie oparte na drzewach) lepiej radzą sobie z samodzielną selekcją istotnych atrybutów, a zewnętrzny selektor mógł usunąć informacje istotne dla specyficznych modeli z portfolio.

Eksperyment 2: One Hot Encoder vs Target Encoder
Porównaliśmy skuteczność dwóch metod kodowania zmiennych kategorycznych w ramach preprocessingu.

- Target Encoder: Balanced Accuracy = 0.851
- One Hot Encoder: Balanced Accuracy = 0.832
- **Wnioski:** Target Encoder okazał się skuteczniejszy od One Hot Encoding. Target Encoder lepiej radzi sobie ze zmiennymi o wysokiej liczbie unikalnych wartości (high cardinality) i nie zwiększa wymiarowości danych tak drastycznie jak OHE, co pozytywnie wpływa na czas i stabilność trenowania modeli.

Eksperyment 3: Liczba Modeli w Ensemble
Zbadano wpływ liczby modeli wybieranych do finalnego ensemble na jakość predykcji.

- Top 3 modele: Balanced Accuracy = 0.844
- Top 5 modeli: Balanced Accuracy = 0.851
- Top 10 modeli: Balanced Accuracy = 0.849
- **Wnioski:** Optymalna liczba to 5 modeli. Mniejsza liczba (3) ogranicza różnorodność, większa (10) może nieznacznie obniżać jakość. Top 5 stanowi dobry kompromis między różnorodnością a stabilnością.

Eksperyment 4: Stacking vs Voting
W celu wyboru optymalnej metody ensemblingu porównano Stacking oraz Voting Classifier.

- Stacking: Balanced Accuracy = 0.851
- Voting Classifier: Balanced Accuracy = 0.822.
- **Wnioski:** Dla naszego systemu stacking osiąga lepsze wyniki, więc ostatecznie wybraliśmy go.

4 Wyniki

Jak wcześniej, jako metrykę wykorzystaliśmy balanced accuracy.

dane	nasze modele	przykładwe modeli
Titanic	0.851	0.798
Spaceship Titanic	0.787	0.780
Bank Marketing	0.881	0.859
Adults	0.867	0.846

Podsumowując wyniki tego projektu, stwierdzamy, że udało nam się dostarczyć kompletny i funkcjonalny system, który spełnia następujące założenia:

- zapewnia powtarzalność wyników,
- wykorzystuje stacking z najlepszymi modelami,
- ma zróżnicowane modele,
- umożliwia uruchamianie systemu na GPU i CPU,
- wspiera równoległe trenowanie modeli,
- wyniki osiągane przez nasze modele są znacząco lepsze niż w przypadku modeli przykładowych.