Mini-AutoML dla danych tabelarycznych

Martyna Sadowska, Zuzanna Zalewska, Aleksandra Zawadka

Wstęp

Celem projektu jest implementacja uproszczonego systemu Mini-AutoML przeznaczonego do rozwiązywania zadań binarnej klasyfikacji na dowolnych zbiorach danych tabelarycznych. Kluczowym założeniem projektu jest wykorzystanie wcześniej zdefiniowanego portfolio modeli, obejmującego różnorodne algorytmy uczenia maszynowego wraz z ustalonymi konfiguracjami hiperparametrów. System dokonuje automatycznej selekcji najlepszego modelu lub zestawu modeli na podstawie danych treningowych, a następnie umożliwia zastosowanie metod zespołowych (ensemblingu) w celu poprawy jakości predykcji.

Zbiory danych

W celu przeprowadzenia etapu wstępnej selekcji modeli oraz oceny ich zachowania na zróżnicowanych problemach klasyfikacji binarnej wykorzystano zestaw dziesięciu publicznie dostępnych zbiorów danych pochodzących z platformy OpenML. Zbiory te zostały dobrane w taki sposób, aby zapewnić możliwie dużą różnorodność pod względem liczby obserwacji, liczby cech czy rodzaju zmiennych. Wykorzystane zbiory danych:

1. dresses-sales – zbiór danych z zmiennymi kategorycznymi opisującymi cechy sukienek. Celem jest przewidzenie, czy dana sukienka zostanie zarekomendowana,
2. credit-g – zbiór służący do oceny ryzyka kredytowego na podstawie cech demograficznych i finansowych. Zadaniem modelu jest klasyfikacja klientów jako wiarygodnych lub niewiarygodnych kredytowo,
3. phoneme – zbiór danych akustycznych służący do rozróżniania samogłosek nosowych i ustnych. Każda obserwacja jest opisana pięcioma cechami odpowiadającymi amplitudom pierwszych harmonicznych sygnału,
4. ozone-level-8hr – zbiór meteorologiczny zawierający pomiary pogodowe i atmosferyczne, takie jak temperatura, wilgotność czy prędkość wiatru. Sprawdzamy, w jakim przypadkach przekroczono dopuszczalny poziom ozonu,
5. steel-plates-fault – dane przemysłowe opisujące wady płyt stalowych. Używane do binarnej klasyfikacji defektów,
6. blood-transfusion-service-center – zbiór danych dotyczący historii donacji krwi, gdzie chcemy przewidzieć czy doszło do ponownej donacji,
7. monks-problems-2 – zbiór danych logicznych, w którym zmienna docelowa jest określona przez regułę logiczną mówiącą, że dokładnie dwie z sześciu cech przyjmują wartość równą 1,
8. qsar-biodeg – zbiór chemiczny z deskryptorami molekularnymi; celem jest predykcja biodegradowalności związków,
9. eeg-eye-state – pomiary aktywności mózgu za pomocą czapki EEG. Celem jest przewidzenie na podstawie 14 sygnałów z różnych miejsc na głowie, czy oczy osoby były otwarte czy zamknięte w danym momencie,
10. jm1 – zbiór danych z inżynierii oprogramowania zawierający metryki kodu źródłowego. Celem jest przewidzenie, czy dany moduł zawiera defekty.

Preprocessing

Proces wstępnego przetwarzania danych został zaprojektowany w sposób uniwersalny, tak aby mógł być automatycznie stosowany do każdego z rozpatrywanych zbiorów danych tabelarycznych, niezależnie od liczby cech, ich typów czy obecności brakujących wartości. Dla każdego zbioru danych preprocessing był wykonywany niezależnie, na podstawie struktury danych wejściowych, bez ręcznej ingerencji użytkownika. Na podstawie typów danych w ramce danych cechy zostały podzielone na numeryczne, kategoryczne oraz `bool`. Taki podział umożliwił zastosowanie odrębnych strategii przetwarzania dla różnych typów zmiennych.

Dla zmiennych numerycznych brakujące wartości były uzupełniane za pomocą imputacji medianą, następnie były standaryzowane za pomocą `StandardScaler`. W przypadku zmiennych kategorycznych brakujące wartości uzupełniano

najczęściej występującą kategorią. Zaś ich przetwarzanie zależało od rodzaju wykorzystywanego modelu. Dla algorytmów drzewiastych (Random Forest, Gradient Boosting, XGBoost, LightGBM oraz CatBoost) zastosowano kodowanie porządkowe `OrdinalEncoder`, ponieważ dobrze współpracuje z ich sposobem podejmowania decyzji i pozwala zachować prostotę reprezentacji cech. W porównaniu do kodowania one-hot podejście to ogranicza wzrost wymiarowości danych, co przekłada się na krótszy czas trenowania modeli. W przypadku wystąpienia nieznanych kategorii w zbiorze testowym przypisywana była im wartość -1. Dla modeli niedrzewiastych, takich jak regresja logistyczna czy k-NN, zmienne kategoryczne były kodowane metodą one-hot encoding przy użyciu `OneHotEncoder`. Dla kodowania one-hot zastosowano mechanizm ignorowania nieznanych kategorii pojawiających się w zbiorze testowym.

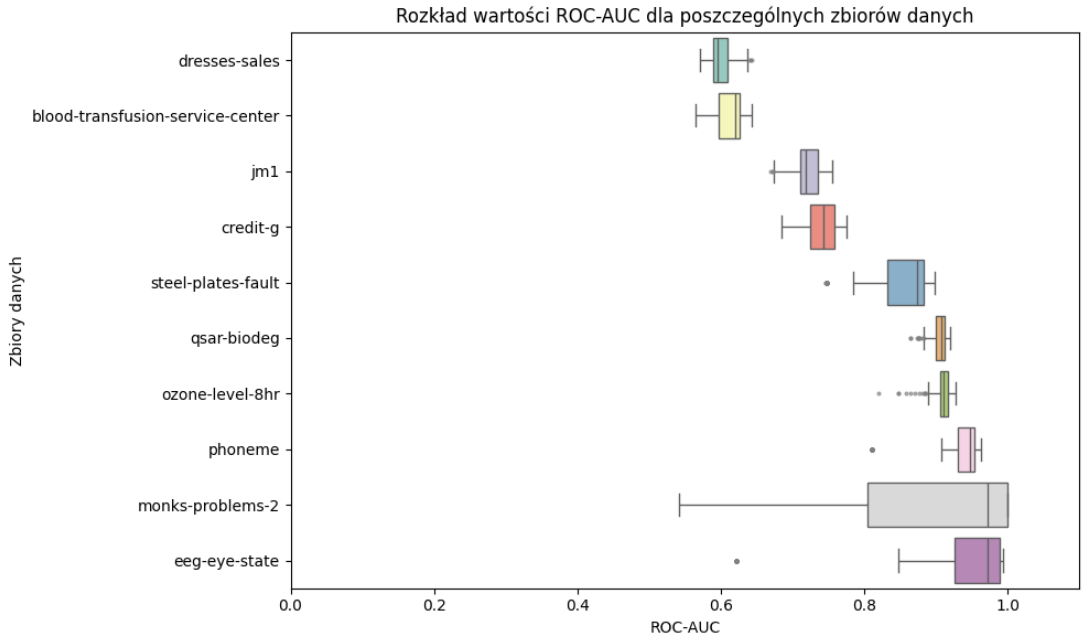
Całość preprocessingu była realizowana przy użyciu `ColumnTransformer`, który umożliwiał równoległe przetwarzanie cech numerycznych i kategorycznych w ramach jednego spójnego potoku. Zaproponowany schemat preprocessingu był stosowany konsekwentnie dla wszystkich analizowanych zbiorów danych.

Opis budowy portfolio modeli

Wykorzystano siedem modeli uczenia maszynowego, reprezentujących różne podejścia do klasyfikacji:

- Logistic Regression
- Random Forest
- Gradient Boosting
- k-Nearest Neighbors
- XGBoost
- LightGBM
- CatBoost

Dla każdego modelu ustalono siatkę hiperparametrów (tabela (4)), z której losowano 30 różnych konfiguracji. Każda konfiguracja była następnie przetrenowana na każdym zbiorze danych, a do oceny jakości działania użyto metryki ROC-AUC, co daje razem 2100 wyników. Metryka ROC-AUC została wybrana ze względu na jej zalety w porównywaniu klasyfikatorów. Dzięki temu możliwe było rzetelne zestawienie wyników między różnymi modelami i zbiorami danych bez konieczności dodatkowego skalowania, co czyni ją odpowiednią miarą do oceny jakości klasyfikacji. Przed trenowaniem każdego modelu wszystkie dane były przetwarzane w jednolity sposób za pomocą preprocessingu opisanego w poprzedniej sekcji. Ponadto, na wykresie (1) został przedstawiony rozkład wartości ROC-AUC uzyskanych przez portfolio modeli na poszczególnych zbiorach danych testowych, ilustrując istotną różnorodność skuteczności klasyfikacji między różnymi typami danych. Takie podejście miało na celu ograniczenie ryzyka nadmiernego dopasowania portfolio modeli do jednego typu danych oraz zwiększenie uniwersalności projektowanego systemu Mini-AutoML.



Rysunek 1: Rozkład wartości ROC-AUC dla poszczególnych zbiorów danych.

Wybór modeli do portfolio

Z początkowej puli 2100 konfiguracji wyselekcjonowano portfolio 50 modeli, opierając się na rankingu średniej globalnej metryki ROC AUC. Wskaźnik ten obliczono dla każdego unikalnego zestawu hiperparametrów jako średnią arytmetyczną wyników uzyskanych na wszystkich analizowanych zbiorach danych. Aby zapewnić różnorodność i uniknąć dominacji jednej rodziny algorytmów, zastosowano limit maksymalnie 8 reprezentantów dla każdego typu modelu. W ten sposób do portfolio weszło po 8 modeli: CatBoost, LightGBM, XGBoost, RandomForest, GradientBoosting, KNN oraz 2 modele LogisticRegression. W tabelach 5–11 przedstawiono wszystkie modele w portfolio, wraz z ich klasami oraz dokładnymi parametrami.

Opis modelu

Model Mini-AutoML został zaprojektowany jako uniwersalny system automatycznej selekcji modeli dla zadań binarnej klasyfikacji na danych tabelarycznych. Podczas inicjalizacji system przyjmuje zbiór wcześniej zdefiniowanych modeli wraz z ich konfiguracjami. Proces uczenia rozpoczyna się od etapu preprocessingu, w którym dane są automatycznie przygotowywane w opisany wcześniej sposób. Selekcja modeli realizowana jest z wykorzystaniem 5-krotnej walidacji krzyżowej ze stratyfikacją. Dla każdej konfiguracji modelu obliczana jest średnia wartość metryki balanced accuracy, która stanowi podstawowe kryterium oceny jakości klasyfikacji. Dodatkowo, dla każdej konfiguracji rejestrowany był całkowity czas obliczeń potrzebny na przeprowadzenie walidacji krzyżowej, obejmujący preprocessing oraz trenowanie modelu. Pozwoliło to na porównanie modeli nie tylko pod względem jakości predykcji, lecz również efektywności obliczeniowej. Na podstawie uzyskanych wyników wybierany jest model o najwyższej średniej wartości balanced accuracy, który następnie trenowany jest na pełnym zbiorze treningowym.

Testowanie ensemblingu

W testowym modelu Mini-AutoML zaimplementowano opcjonalny mechanizm ensemblingu oparty na metodzie głosowania. W tym wariancie, zamiast wyboru modelu o najlepszym średnim balanced accuracy, wybieranych było 5 najlepszych modeli, które następnie były używane w modelu Votingowym z miękkim głosowaniem. Skuteczność tego podejścia została oceniona na zbiorach danych wykorzystanych wcześniej w procesie budowy portfolio. Dla każdego zbioru porównano wartości balanced accuracy uzyskane dla najlepszego pojedynczego modelu oraz przez model Votingowy. Otrzymane wartości przedstawione są w tabeli 1.

Nazwa zbioru	**Najlepszy model**	**Bal. acc. bez votingu**	**Bal. acc. z votingiem**
dresses-sales	logreg_48	0.578	0.592
credit-g	logreg_49	0.677	0.668
phoneme	catboost_12	0.873	0.878
ozone-level-8hr	knn_42	0.781	0.783
steel-plates-fault	catboost_0	1.000	1.000
blood-transfusion	logreg_49	0.627	0.667
monks-problems-2	catboost_0	1.000	1.000
qsar-biodeg	xgboost_10	0.830	0.848
eeg-eye-state	catboost_11	0.962	0.963
jm1	knn_40	0.687	0.687

Tabela 1: Balanced accuracy bez i z votingiem dla poszczególnych zbiorów danych

Po analizie otrzymanych wyników postanowiono wprowadzić powyższy mechanizm ensemblingu jako stały element systemu Mini-AutoML. Jednocześnie, aby uniknąć sytuacji, w której model Votingowy osiąga niższe wyniki niż najlepszy model indywidualny, zastosowano hybrydowe kryterium wyboru modelu końcowego. Ostatecznie główny model jest wybierany po wyższym wyniku balanced accuracy spośród najlepszego modelu pojedynczego oraz modelu Votingowego.

Opis techniczny implementacji Mini-AutoML

System Mini-AutoML został zaimplementowany w postaci klasy `MiniAutoML`, w której możemy wyróżnić poszczególne metody:

- `load_models_from_json`: wczytanie modeli wraz z ich parametrami i ustawienie ziarna losowości dla reprodukowalności wyników,
- `get_preprocessor`: utworzenie pipeline opisanego wcześniej preprocessingu

- `fit`: przeprowadzenie 5-krotnej walidacji krzyżowej dla każdego modelu, obliczenie średniego balanced accuracy i czasu wykonania, a następnie stworzenie modelu Votingowego z pięcioma najlepszymi modelami oraz wybór ostatecznego modelu końcowego na podstawie wyższej skuteczności,
- `predict`: przewidywanie klas dla nowych danych,
- `predict_proba`: przewidywanie prawdopodobieństw przynależności do klasy pozytywnej.

Wyniki

Zestawienie wyników poszczególnych konfiguracji modeli, wraz z odpowiadającymi im czasami obliczeń dla dostarczonego przykładowego zbioru danych przedstawiono w tabeli 2. Oprócz pełnego zestawienia wyników dla poszczególnych konfiguracji modeli, w tabeli 3 zaprezentowano również zagregowane wyniki według typu algorytmu. Zestawienie to pozwala na ogólne porównanie skuteczności oraz kosztu obliczeniowego poszczególnych rodzin modeli, niezależnie od konkretnych ustawień hiperparametrów.

Model	Bal. acc.	Czas [s]	Model	Bal. acc.	Czas [s]
catboost_0	0.5410	7.96	xgboost_20	0.5239	1.98
lightgbm_1	0.5263	4.19	xgboost_21	0.5282	2.01
xgboost_2	0.5385	0.88	rf_22	0.5439	46.18
rf_3	**0.5519**	**16.15**	lightgbm_23	0.5335	0.71
rf_4	**0.5532**	**5.80**	xgboost_24	0.5335	0.73
xgboost_5	0.5274	1.41	lightgbm_25	0.5169	1.87
catboost_6	0.5415	24.60	xgboost_26	0.5206	2.70
lightgbm_7	0.5333	2.78	**rf_27**	**0.5506**	**6.56**
catboost_8	0.5236	7.02	gb_28	0.5179	6.05
catboost_9	0.5444	2.01	gb_29	0.5262	8.39
xgboost_10	0.5217	1.45	rf_30	0.5457	10.08
catboost_11	0.5326	35.06	rf_31	0.5467	11.34
catboost_12	0.5455	19.73	rf_32	0.5448	5.30
catboost_13	0.5331	7.80	gb_33	0.5278	12.32
catboost_14	0.5395	9.47	rf_34	0.5470	29.51
xgboost_15	0.5257	2.84	gb_35	0.5220	6.10
lightgbm_16	0.5185	1.32	gb_36	0.5209	8.84
lightgbm_17	0.5143	1.90	gb_37	0.5284	3.59
lightgbm_18	0.5321	3.35	gb_38	0.5146	7.20
lightgbm_19	0.5182	2.64	gb_39	0.5241	15.92
knn_40	0.5407	0.12	**knn_44**	**0.5493**	**0.09**
knn_41	0.5470	0.10	knn_45	0.5468	0.16
knn_42	**0.5500**	**0.09**	**knn_46**	**0.5493**	**0.07**
knn_43	0.5469	0.08	knn_47	0.5435	0.06
logreg_48	0.5338	0.10	logreg_49	0.5364	0.08

Tabela 2: Wyniki 5-krotnej walidacji krzyżowej dla testowanych modeli. Wyróżnionych zostało 5 najlepszych wyników balanced accuracy.

Model	Śr bal. acc.	Śr czas [s]
Random Forest	0.5480	16.37
KNN	0.5467	0.10
CatBoost	0.5377	14.21
LogReg	0.5351	0.09
XGBoost	0.5274	1.75
LightGBM	0.5241	2.35
Gradient Boost	0.5227	8.55

Tabela 3: Wyniki zagregowane według typu modelu.

Wnioski

Analiza wyników przedstawionych w tabelach 2 i 3 pozwala wysunąć następujące wnioski co do wybieranych przez nasz system modeli:

1. Spośród wszystkich testowanych konfiguracji najwyższe wartości średniego balanced accuracy uzyskały modele Random Forest oraz k-NN, osiągając poziom około 0.55.
2. Analiza czasu obliczeń wskazuje, że modele k-NN oraz Logistic Regression są wyjątkowo szybkie, co czyni je korzystnymi w zastosowaniach wymagających szybkiej predykcji. W przypadku modeli drzewiastych, takich jak Random Forest czy CatBoost, czas treningu jest znacznie większy.
3. Zagregowane wyniki pokazują, że Random Forest osiąga najwyższą średnią skuteczność. Modele k-NN osiągają niemal równoważną skuteczność przy minimalnym koszcie czasowym. W przypadku CatBoost oraz Logistic Regression obserwujemy umiarkowaną skuteczność, przy czym CatBoost wymaga znacznie więcej czasu. Najniższe wartości średniego balanced accuracy uzyskują XGBoost, LightGBM oraz Gradient Boosting.

Załącznik

Algorytm	Hiperparametr	Typ	Dolna granica	Górna granica
Logistic Regression	*C*	ciągły	0.001	10
	l1_ratio	ciągły	0	1
Random Forest	*n_estimators*	całkowity	100	1000
	max_depth	całkowity	5	40
	min_samples_split	całkowity	2	20
	min_samples_leaf	całkowity	1	10
	max_features	ciągły	0.2	1.0
Gradient Boosting	*n_estimators*	całkowity	100	1000
	learning_rate	ciągły	0.01	0.31
	max_depth	całkowity	2	5
	subsample	ciągły	0.5	1.0
	max_features	ciągły	0.3	1.0
k-Nearest Neighbors	*n_neighbors*	całkowity	3	50
XGBoost	*n_estimators*	całkowity	100	1500
	max_depth	całkowity	3	10
	learning_rate	ciągły	0.01	0.31
	subsample	ciągły	0.5	1.0
	colsample_bytree	ciągły	0.5	1.0
LightGBM	*n_estimators*	całkowity	100	1500
	learning_rate	ciągły	0.01	0.31
	num_leaves	całkowity	20	200
	max_depth	całkowity	3	15
	min_child_samples	całkowity	5	100
	subsample	ciągły	0.5	1.0
	colsample_bytree	ciągły	0.5	1.0
CatBoost	*iterations*	całkowity	100	1500
	depth	całkowity	3	10
	learning_rate	ciągły	0.01	0.31
	l2_leaf_reg	ciągły	1	11
	bagging_temperature	ciągły	0	1

Tabela 4: Zakresy hiperparametrów wykorzystane podczas losowego przeszukiwania przestrzeni konfiguracji modeli. W tabeli uwzględniono tylko parametry liczbowe i całkowite. Dodatkowo w modelach występowały parametry kategoryczne i stałe: w LogisticRegression: *penalty* $\in$ {l1, l2, elasticnet}, *solver*=saga, *max_iter*=2000; w KNN: weights $\in$ {uniform, distance}, $p \in \{1,2\}$.

Model ID	iterations	depth	learning_rate	l2_leaf_reg	bagging_temperature
catboost_0	435	7	0.0519	2.9991	0.0074
catboost_6	1049	8	0.0684	5.6672	0.0438
catboost_8	793	6	0.1243	4.0151	0.6303
catboost_9	228	6	0.0291	9.2763	0.6317
catboost_11	1489	8	0.0802	1.5830	0.2814
catboost_12	1494	7	0.1319	9.1402	0.1670
catboost_13	331	8	0.1164	10.7070	0.8931
catboost_14	715	7	0.1513	6.6524	0.7650

Tabela 5: CatBoostClassifier – użyte hiperparametry

Model ID	n_estimators	lr	num_leaves	max_depth	min_child	subsample	colsample
lightgbm_1	278	0.0707	102	11	34	0.7896	0.5884
lightgbm_7	351	0.1090	108	10	18	0.5517	0.7938
lightgbm_16	487	0.1477	160	9	59	0.5699	0.5573
lightgbm_17	296	0.2050	125	10	13	0.7079	0.5208
lightgbm_18	1213	0.0490	54	11	50	0.6135	0.8349
lightgbm_19	730	0.1764	60	13	11	0.9138	0.8158
lightgbm_23	248	0.2267	36	11	48	0.5806	0.7505
lightgbm_25	465	0.1448	149	11	30	0.5482	0.9513

Tabela 6: LightGBMClassifier – użyte hiperparametry

Model ID	n_estimators	max_depth	lr	subsample	colsample
xgboost_2	228	6	0.0291	0.9138	0.8158
xgboost_5	331	8	0.1164	0.9853	0.9466
xgboost_10	224	8	0.1417	0.9293	0.8487
xgboost_15	950	5	0.0363	0.5590	0.9809
xgboost_20	793	6	0.1243	0.6508	0.8151
xgboost_21	715	7	0.1513	0.7826	0.8825
xgboost_24	127	8	0.2258	0.7160	0.8137
xgboost_26	1049	8	0.0684	0.7334	0.5219

Tabela 7: XGBoostClassifier – użyte hiperparametry

Model ID	n_estimators	max_depth	min_split	min_leaf	max_features
rf_3	666	29	3	3	0.8658
rf_4	177	29	5	1	0.9805
rf_22	855	20	11	4	0.9414
rf_27	555	24	2	2	0.2916
rf_30	993	34	2	3	0.2466
rf_31	495	28	13	5	0.6522
rf_32	180	32	13	4	0.8869
rf_34	872	33	6	6	0.8065

Tabela 8: RandomForestClassifier – użyte hiperparametry

Model ID	n_estimators	lr	max_depth	subsample	max_features
gb_28	710	0.0684	5	0.7334	0.3307
gb_29	993	0.0802	5	0.5292	0.4970
gb_33	791	0.2514	5	0.6937	0.5018
gb_35	248	0.1164	5	0.9853	0.9252
gb_36	996	0.1319	4	0.9070	0.4169
gb_37	180	0.1417	5	0.9293	0.7882
gb_38	495	0.1513	4	0.7826	0.8355
gb_39	817	0.2095	5	0.8526	0.8465

Tabela 9: GradientBoostingClassifier – użyte hiperparametry

Model ID	n_neighbors	weights	p
knn_40	46	distance	1
knn_41	42	distance	1
knn_42	39	distance	1
knn_43	28	distance	1
knn_44	37	distance	1
knn_45	48	distance	2
knn_46	33	distance	2
knn_47	27	distance	2

Tabela 10: KNeighborsClassifier – użyte hiperparametry

Model ID	C	penalty	solver	l1_ratio	max_iter
logreg_48	0.9428	l1	saga	0.9756	2000
logreg_49	1.3002	l1	saga	0.4757	2000

Tabela 11: LogisticRegression – użyte hiperparametry