Politechnika Warszawska

Wydział Matematyki
i Nauk Informacyjnych

Mini-AutoML - Raport z projektu

Mateusz Brochocki
Aleksander Karch
Adrian Krzyżanowski

Warszawa, Polska

26 stycznia 2026

1 Cel projektu

Celem projekt jest stworzenie miniaturowej wersji systemu AutoML, której zadaniem będzie automatyczne rozwiązanie problemu klasyfikacji binarnej. W systemie zostanie wykorzystana koncepcja portfolio, a więc gotowych modeli z wcześniej zoptymalizowanymi hiperparametrami. System będzie zawierał 50 modeli, a ponadto użyte zostaną techniki preprocessingu oraz ensemblingu w celu zwiększenia jakości predykcji.

2 Wybór modeli składowych

Podczas wyboru modeli do portfolio skupiliśmy się na modelach drzewiastych, liniowych, a także na metodzie wektorów podpierających (SVM) oraz metodzie najbliższych sąsiadów (kNN). W przypadku modeli drzewiastych zajęliśmy się modelem CatBoost, który jest jednym z lepszych, obecnie dostępnych, modeli tego typu, używanym w wielu pracach jako źródło dobrych benchmarkowych wyników np. Prior Labs Team, *TabPFN-2.5: Advancing the State of the Art in Tabular Foundation Models*, 2025. Modele liniowe były zaś reprezentowane przez regresję logistyczną z regularyzacją typu elastic net.

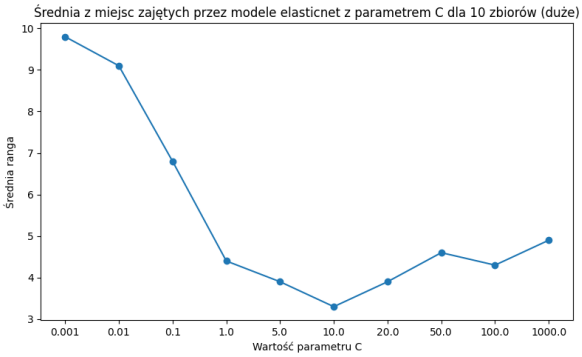
2.1 Wybór CatBoostów

CatBoosty zostały wybrane za pomocą optymalizacji TPE oraz pakietu `optuna`. Wybranych zostało 45 catboostów, po 3 na każdy testowany zbiór danych: mały (10 drzew), mniej mały (20 drzew), spory (200 drzew). Dodatkowo, został dodany pojedynczy model z 1000 drzew, również znaleziony przy pomocy optuny. Część z wybranych modeli została następnie ręcznie usunięta ze względu na zbyt długi czas wykonania (przynajmniej jak na nasze potrzeby). W ten sposób uzyskaliśmy finalnie 38 modeli typu CatBoostClassifier.

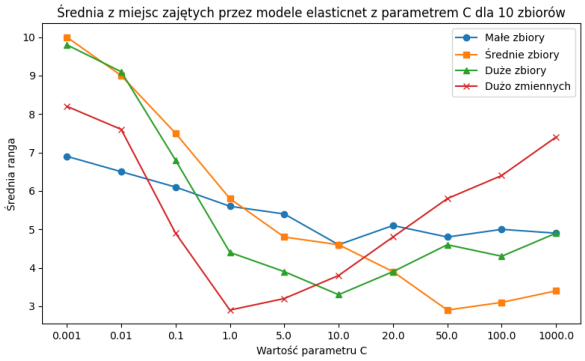
2.2 Wybór pozostałych modeli

W przypadku wyboru pozostałych modeli skupiliśmy się na optymalizacji dwóch lub jednego parametru. Dla regresji były to parametr regularyzacji C oraz parametr *l1_ratio*. Dla metody kNN parametr *n_neighbors* związany z liczbą sąsiadów, a także parametr p określający normę, w której mierzona jest odległość między sąsiadami. W przypadku SVM optymalizowaliśmy parametr regularyzacji C. Użyliśmy do tego 40 zbiorów danych z OpenML'a, które to podzieliliśmy na 4 kategorię po 10 zbiorów każda. Pierwsza z nich to zbiory małe do 700 obserwacji, druga to zbiory średnie do 5000 obserwacji, zaś trzecia to zbiory duże powyżej 5000 obserwacji. Oddzielną kategorią były zbiory z duża liczbą zmiennych objaśniających (powyżej 50 zmiennych), do której trafiały zbiory bez względu na rozmiar danych. Dla każdej kategorii stworzyliśmy po jednym modelu każdego rodzaju, co dało nam w sumie 12 różnych modeli. Sposób poszukiwania optymalnych hiperparametrów był bardzo podobny dla każdego modelu i każdej kategorii. Polegał on na przeszukaniu szerokiego i naturalnego zakresu wartości dla jednego parametru za pomocą *GridSearchCV* (w przypadku dwóch był to parametr istotniejszy, a wybór jego odbywał się na podstawie poprzedniego projektu dotyczącego tunowalności), a następnie określeniu, w jakim zakresie są uzyskiwane najlepsze wyniki i tam dokonanie dokładniejszego poszukiwania optymalnej wartości prze użyciu *RandomSearchCV*. W ten sposób uzyskiwaliśmy 10 optymalnych wartości hiperparametru, po jednej dla każdego zbioru danych z danej kategorii. Następnie wyniki te agregowaliśmy za pomocą średniej i mody, żeby na koniec policzyć średnią z tych dwóch wartości i w ten sposób uzyskać optymalną wartość hiperparametru dla danej kategorii. W sytuacji, gdy występował drugi hiperparametr powtarzaliśmy rozumowanie z ustaloną wartością pierwszego hiperparametru uzyskaną w poprzednim kroku. Mogła zdarzyć się sytuacja, że nie było określonego przedziału wartości hiperparametru, w którym następowałaby znacząca

poprawa, a więc hiperparametr odznaczał się mało tunowalnością. Wtedy nie dokonywaliśmy wewnętrznego przeszukiwania, a wartość optymalną hiperparametru ustalaliśmy na podstawie wyników z pierwszego przeszukiwania.

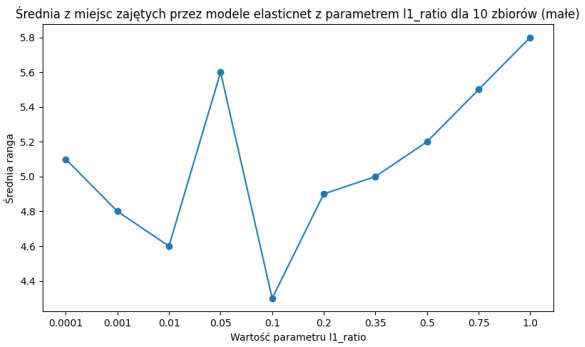


(a) Przeszukiwanie ustalonej wstępnej siatki parametrów dla parametru C w modelu elastic net dla 10 dużych zbiorów. Na osi pionowej znajduje się średnia z miejsc (1-10) zajętych przez dany parametr, co oznacza, że na przykład model z wartością $C = 0.001$ okazywał się prawie zawsze najgorszy wśród 10 ustalonych hiperparametrów dla każdego z 10 zbiorów. Widzimy, że optymalne możliwym zakresem do potencjalnego dokładniejszego przeszukania jest przedział $[5, 20]$.

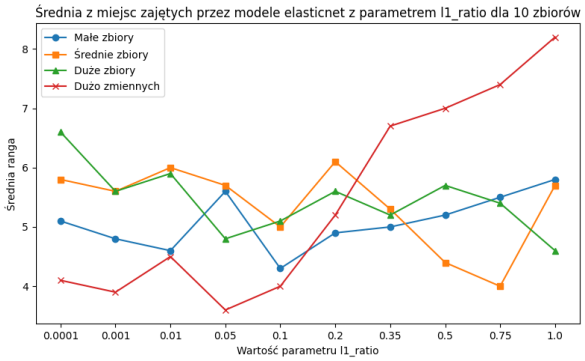


(b) Przeszukiwanie ustalonej wstępnej siatki parametrów dla parametru C w modelu elastic net dla każdej z czterech kategorii zbiorów. Widać tutaj różnicę w tunowalności parametru C dla różnych kategorii zbiorów. Na przykład dla małych zbiorów tunowalność parametru C jest najmniejsza, choć nadal istotna. Ponadto możemy zauważyć, że zakresy dokładniejszego przeszukania są różne i na przykład dla średnich zbiorów są znacznie większe niż dla pozostałych przypadków.

Rysunek 1: Wizualizacja pierwszego kroku procedury poszukiwania optymalnych hiperparametrów dla parametru C w modelu elastic net.



(a) Przeszukiwanie ustalonej wstępnej siatki parametrów dla parametru *l1_ratio* w modelu elastic net dla 10 małych zbiorów. Widzimy tutaj, że tunowalność tego parametru jest na bardzo niskim poziomie (bardzo podobny średni wynik dla każdej wartości hiperparametru), dlatego poprzestajemy na ustaleniu jako optymalną wartość hiperparametru wartości *l1_ratio* $= 0.1$.



(b) Przeszukiwanie ustalonej wstępnej siatki parametrów dla parametru *l1_ratio* w modelu elastic net dla każdej z czterech kategorii zbiorów. Widzimy tutaj, że tylko dla zbiorów z dużą liczbą zmiennych występuje istotna tunowalność hiperparametrów i tylko dla nich będziemy wykonywać krok drugi naszej optymalizacji, a więc dokładniejsze przeszukanie przedziału $[0.0001, 0.1]$.

Rysunek 2: Wizualizacja pierwszego kroku procedury poszukiwania optymalnych hiperparametrów dla parametru *l1_ratio* w modelu elastic net.

3 KGBClassifier - działanie

3.1 Preprocessing

Przed selekcją najlepszego modelu, nasz system KGBClassifier wykonuje oddzielny preprocessing dla CatBoostów i pozostałych modeli. Jeśli chodzi o Catboosty, to wykonujemy uzupełnianie braków dla zmiennych numerycznych i kategorycznych przy pomocy, odpowiednio, średniej i najczęściej występującej wartości. W przypadku reszty modeli dodatkowo wykonujemy skalowanie za pomocą MinMax oraz procedurę One-Hot Encoding. Uzupełnienie braków wykonujemy tak samo jak dla CatBoostów.

3.2 Selekcja modeli

Wybór najlepszego modelu podzieliliśmy na dwie części. W części pierwszej wybieramy najlepszy model spośród CatBoostów, a w drugiej najlepszy spośród pozostałych modeli. Jeśli chodzi o CatBoosty, to przyjęliśmy heurystyczne ograniczenie czasowe - trenujemy kolejne modele dopóki nie zostanie nam mniej niż 6 minut z 20 minutowego całościowego ograniczenia czasowego. Każdy model trenowany jest na tym samym splicie, a następnie oceniany według tej samej miary, czyli Balanced Accuracy.
W przypadku pozostałych modeli przejęliśmy heurystyczne ograniczenie czasowe - trenujemy modele dopóki nie zostanie nam mniej niż 4 minuty. Tutaj uwzględniliśmy kategorie wielkości zbiorów danych, o których była mowa w poprzednim rozdziale. To znaczy, najpierw trenujemy modele dopasowane do wielkości danych, a jeśli starczy czasu, to trenujemy pozostałe modele. W każdej iteracji pętli trenujemy w sumie 3 modele naraz (jedna regresja, jeden SVM, jeden kNN) na tym samym splicie. Następnie oceniamy każdy z modeli za pomocą miary Balanced Acccuracy.
W kolejnym kroku wybieramy jeden najlepszy model spośród CatBoostów i jeden najlepszy spośród pozostałych modeli (najlepszy, czyli z najwyższym wynikiem Balanced Accuracy). Podsumowując, cała metoda selekcji odbywa się za pomocą brute-force z ograniczeniem czasowym. W przypadku modeli niebędących CatBoostami wykorzystujemy heurystykę opartą na rozmiarze danych. Takie podejście być może nie jest najbardziej stabilne, jednak daje sporą oszczędność czasową.

3.3 Ensembling

KGBClassifier wykorzystuje dwie metody ensemblingu - Stacking i Voting. System w ogóle nie wykona ensemblingu, jeśli ma mniej niż 2 minuty z budżetu czasowego. Do Stackingu wybieramy trzy losowe modele typu CatBoost (ale bez najlepszego), najlepszy model z pozostałych modeli i losowy model z pozostałych (niebędący najlepszym). Jako finalny model w Stackingu wybieramy najlepszy CatBoost. Następnie trenujemy otrzymany Stacking i obliczamy miarę Balanced Accuracy. Przy okazji mierzymy czas wykonywania się Stackingu.
Jeśli zostało nam więcej czasu niż czas, w którym wykonywał się Stacking, to rozpoczynamy trenowanie Votingu typu hard. Ogólnie do Votingu wykorzystujemy najlepszy model typu CatBoost, dwa losowe CatBoosty, najlepszy model z pozostałych (nie-Catboostów) oraz losowy model z pozostałych (niebędący najlepszym). Podobnie jak wcześniej, trenujemy otrzymany Voting i liczymy miarę. Jeśli została nam odpowiednia ilość czasu, to wykonujemy analogiczną procedurę dla Votingu typu soft.

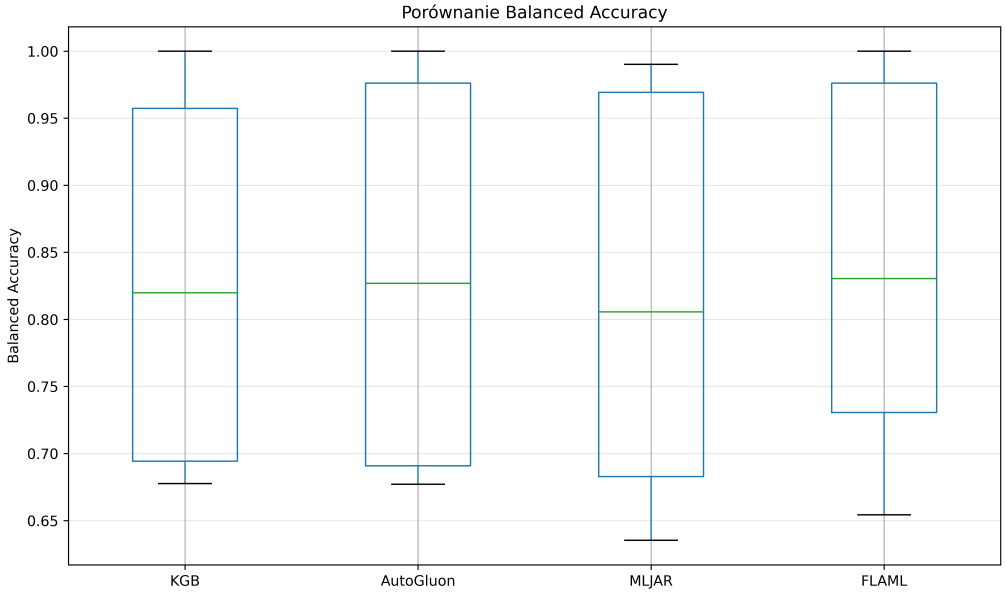
Na koniec z 5 otrzymanych modeli, czyli najlepszego CatBoosta, najlepszego nie-Catboosta oraz trzech opisanych powyżej ensembli, wybieramy ten najlepszy, czyli mający największą wartość Balance Accuracy i to on jest zwracany przez metodę *fit*.

4 Eksperyment i wnioski

W części eksperymentalnej naszego projektu porównaliśmy wyniki otrzymane za pomocą KGBClassifier z wynikami otrzymanymi przez inne pakiety AutoMl, czyli AutoGluon, MLJAR Supervised i FLAML. W tym celu wytrenowaliśmy modele z każdego pakietu na 5 zbiorach danych, a następnie porównaliśmy ze sobą miary Balanced Accuarcy. Przy uczeniu każdego zbioru danych ustawiliśmy ograniczenie czasowe na 3 minuty. Wyniki eksperymentu można odczytać z poniższej tabeli i wykresu.

Tabela 1: Pomiary Balanced Accuracy dla różnych pakietów AutoML.

Numer zbioru danych	**KGBClassifier**	**AutoGluon**	**MLJAR**	**FLAML**
I	0.6776	0.6908	0.6352	0.6542
II	1	1	0.9902	1
III	0.8199	0.8268	0.8056	0.8305
IV	0.9573	0.9762	0.9692	0.9762
V	0.6943	0.677	0.6828	0.7306



Rysunek 3: Wykres przedstawia rozkłady wyników miary Balanced Accuracy dla poszczególnych pakietów AutoML w postaci wykresów skrzynkowych.

Jak mozna zauważyć wyniki uzyskane za pomocą KGBClassifiera nie odbiegają znacząco od wyników osiąganych przez pozostałe pakiety. Co więcej, na podstawie tych 5 zbiorów można powiedzieć, że KGBClassifier radzi sobie przeciętnie trochę lepiej niż pakiet MLJAR, ale za to trochę gorzej od AutoGluona i FLAML. Mediana dla KGBClassifiera jest nieco większa od mediany MLJAR'a, ale odrobinę mniejsza od median dla Autogluona i FLAML. Jeśli chodzi o rozrzuty, to FLAML wygląda najstabilniej, KGB ma rozrzut porównywalny do AutoGluon, a MLJAR ma największy rozrzut. W przypadku zbioru II modele ze wszystkich pakietów najpewniej są przeuczone. Podsumowując, KGBClassifier jednak może być używany jako zamiennik dla pozostałych analizowanych pakietów.