

AutoML - projekt 1

Jan Kwiecień, Filip Mieszkowski, Stanisław Kurzatkowski

1 Wstęp

Celem projektu było przeanalizowanie tunowalności hiperparametrów 3 wybranych algorytmów uczenia maszynowego (u nas xgboost, random forest, MLP) na co najmniej 4 zbiorach danych. Do tunowania modeli należało użyć dwóch technik losowania punktów z siatki hiperparametrów (Uniform Grid Search, BayesSearchCV).

2 Zbiory danych

W naszym projekcie podjęliśmy się problemu klasyfikacji. Wybraliśmy cztery zbiory danych, żeby sprawdzić różne elementy naszego pipeline'u. Są to: **Jannis** (wieloklasowy, głównie dane ilościowe) – średni/duży zestaw, **Car Evaluation** (wieloklasowy, głównie dane jakościowe) – niewielki, **APSFailure w Scania Trucks** – z dużą ilością braków danych i niezrównoważonymi klasami oraz **Covertype** (wieloklasowy, mieszany) – średni/duży zestaw z cechami ilościowymi i jakościowymi. Dzięki takiej mieszance mamy różne rozmiary danych (od małych po duże), 3 zadania wieloklasowe i jedno binarne. To pozwala nam porównywać hiperparametry i algorytmy w naprawdę różnych scenariuszach.

3 Random forest

Pierwszym algorytmem, który testowaliśmy był random forest. Parametry, które wybraliśmy do tunowania to: liczba drzew (50, 100, 150, 200), maksymalna głębokość (5, 10), minimalna liczba próbek do podziału (2, 5), max_features (sqrt, log2). Dla tego algorytmu siatka parametrów jest dość uboga, z powodu ograniczeń obliczeniowych, a ich zakresy zostały ustalone na podstawie artykułów [1], [2] oraz [3]. Ponadto na podstawie artykułu [2] spodziewamy się w przypadku tego algorytmu tunowalność nie będzie wysoka lub w ogóle jej nie zaobserwujemy.

3.1 Wynik RandomSearchCV i BayesSearchCV

W poniższej tabeli zaprezentujemy wyniki poszukiwań parametrów dla metod RandomSearchCV i BayesSearchCV. Dla każdego z nich używaliśmy podwójnej kroswalidacji, a każda metoda korzystała z ośmiu iteracji. Parametry zostały wybrane na podstawie accuracy.

	Random Search	Bayes Search
Jannis	(100, 10, 2, 'sqrt')	(150, 10, 5, 'sqrt')
Cars	(200, 15, 2, 'log2')	(200, 15, 2, 'log2')
APS	(100, 10, 2, 'log2')	(200, 10, 5, 'sqrt')
Covertype	(100, 10, 2, 'sqrt')	(200, 10, 5, 'sqrt')

Tabela 1: Tabela z optymalnymi parametrami

3.2 Wyznaczanie optymalnej konfiguracji parametrów

Kolejnym etapem eksperymentu będzie wyznaczenie optymalnej konfiguracji hiperparametrów, którą następnie porównamy z defaultowym modelem. Wyniki tej części przedstawia następująca tabela.

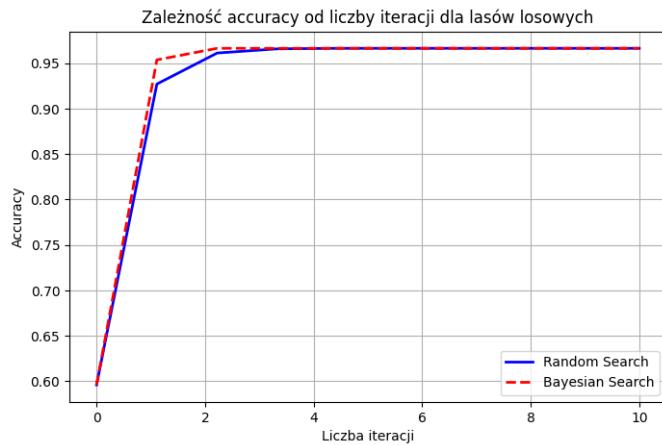
	Jannis	Cars	APS	Covertype
(100, 10, 2, 'sqrt')	0.677	0.946	0.992	0.751
(150, 10, 5, 'sqrt')	0.677	0.944	0.993	0.748
(200, 15, 2, 'log2')	0.692	0.960	0.993	0.813
(100, 10, 2, 'log2')	0.671	0.946	0.993	0.741
(200, 10, 5, 'sqrt')	0.678	0.940	0.993	0.748
default	0.700	0.969	0.994	0.951

Tabela 2: Accuracy otrzymane dla różnych konfiguracji hiperparametrów

Jako model referencyjny wybierzemy ten, którego najmniejsze accuracy było największe. Jest to model z parametrami (200, 15, 2, 'log2'). Niestety, dla każdego zbioru danych wypadł on gorzej. Może to potwierdzać tezę, że lasy losowe są słabo tunowalne lub może należało przetestować większą siatkę parametrów.

3.3 Stabilność metody i porównanie Random Search z Bayes Search

W celu zbadania stabilności obydwu metod stworzyliśmy eksperyment badający zależność accuracy od liczby iteracji (dla zbioru cars).



Rysunek 1: Zależność accuracy od liczby iteracji

Jak widzimy obie metody bardzo podobnie zbiegają do podobnej wartości accuracy. W drugiej części porównamy metody Random Search z Bayes Search. Ponownie wybierzemy takie hiperparametry, które odpowiednio wśród obu metod dały najwyższe wyniki dla najmniejszych accuracy. Dla obu metod zestaw hiperparametrów okazał się być identyczny (200, 15, 2, 'log2'). W związku z tym, przy takiej małej siatce hiperparametrów obydwie metody okazały się tak samo skuteczne.

4 XGBoost

Następny przetestowany klasyfikator to **XGBClassifier** z biblioteki **XGBoost**. Przeszukiwana przez algorytmy RandomSearch oraz BayesSearch przestrzeń hiperparametrów przedstawiona jest w tabeli 9.

4.1 Wynik RandomSearchCV i BayesSearchCV

W tabeli 10 umieszczone są wyniki poszukiwań parametrów dla metod RandomSearchCV i BayesSearchCV. Każdej konfiguracji (lambda) odpowiada jedna z dwóch metod i jeden z czterech zbiorów wykorzystanych do analizy, nazwy konfiguracji mają format **metoda_zbiór** (metoda użyta do znalezienia konfiguracji oraz zbiór na podstawie którego szukano konfiguracji). Każda metoda wykonała 30 iteracji, mając do dyspozycji 80% całego zbioru. Każdy z 4 zbiorów został podzielony dokładnie raz, w taki sposób, żeby żadna instancja ze zbioru testującego (pozostałe 20%) dowolnego zbioru danych nie wyciekła do procesu optymalizacji hiperparametrów modelu. Zbiór uczący był wykorzystany do poszukiwań kombinacji hiperparametrów z kroswalidacją wynoszącą 4, dzięki czemu został on całkowicie wyeksploatowany w celach uczenia i walidacji modelu w etapach pośrednich.

lambda_name	accuracy	f1_macro
Bayes_aps	0,710217	0,555413
Bayes_car	0,717382	0,546090
Bayes_covertype	0,705523	0,538780
Bayes_jannis	0,720511	0,559559
Random_aps	0,714575	0,541581
Random_car	0,713752	0,548852
Random_covertype	0,720810	0,560636
Random_jannis	0,721036	0,551488

Tabela 3: Wyniki poszczególnych konfiguracji obliczone, jako *minimum* z wyników na wszystkich 4 zbiorach

lambda_name	accuracy	f1_macro
Bayes_aps	0,710217	0,555413
Bayes_car	0,717382	0,546090
Bayes_covertype	0,705523	0,538780
Bayes_jannis	0,720511	0,559559
Random_aps	0,714575	0,541581
Random_car	0,713752	0,548852
Random_covertype	0,720810	0,560636
Random_jannis	0,721036	0,551488

Tabela 4: Wyniki poszczególnych konfiguracji tylko na zbiorze *jannis*

lambda_name	accuracy	f1_macro
Bayes_aps	0,868725	0,780705
Bayes_car	0,902410	0,838335
Bayes_covertype	0,912560	0,835035
Bayes_jannis	0,881005	0,772784
Random_aps	0,871615	0,782250
Random_car	0,879249	0,807974
Random_covertype	0,900459	0,840515
Random_jannis	0,879552	0,751340

Tabela 5: Wyniki poszczególnych konfiguracji obliczone, jako *średnia* z wyników na wszystkich 4 zbiorach

4.2 Wyznaczanie optymalnej konfiguracji parametrów

Powyżej przedstawione są 3 tabele 345, zestawiające ze sobą wszystkie 8 konfiguracji hiperparametrów w oparciu o metryki *accuracy* oraz *F1*. Metrykę *accuracy* przyjęto, jako ogólny wyznacznik "sprawności" modelu, natomiast *F1* jako miarę pomocniczą dla zbiorów niezabalansowanych takich, jak *jannis*. Kolorem pomarańczowym oznaczona najlepiej oceniona metody, intensywność koloru czerwonego odpowiada 3 najwyższy wynikom.

Wartości liczbowe w przedstawionych tabelach mocno sugerują, iż zbiorom najtrudniejszym do przeprowadzenia zadania klasyfikacji był zbiór *jannis*. Z tego powodu postanowiono wziąć pod uwagę również tabelę z wynikami uśrednionymi, gdyż w innym przypadku wyłonienie optymalnej konfiguracji sprowadziłoby się do znalezienia konfiguracji najlepiej sprawdzającej się dla zbioru *jannis*. Patrząc jednak na Tabele [3] oraz [5] można zauważać, że kombinacja *Random_covertype* zalicza się do obydwu grup najlepiej ocenianych kombinacji hiperparametrów, dzięki czemu staje się ona naturalnym wyborem na kombinację optymalną.

4.3 Porównanie *Random_covertype* z domyślną konfiguracją biblioteki *XGBoost*

Dataset	lambda_name	Accuracy	F1-macro
aps	XGB_default	0.994605	0.921848
car	XGB_default	0.997110	0.989745
covertype	XGB_default	0.872095	0.855053
jannis	XGB_default	0.715292	0.556689
aps	Random_covertype	0.995987	0.940662
car	Random_covertype	0.988439	0.983410
covertype	Random_covertype	0.898445	0.883218
jannis	Random_covertype	0.723771	0.566052
Average	XGB_default	0.894776	0.830834
Average	Random_covertype	0.901661	0.843336

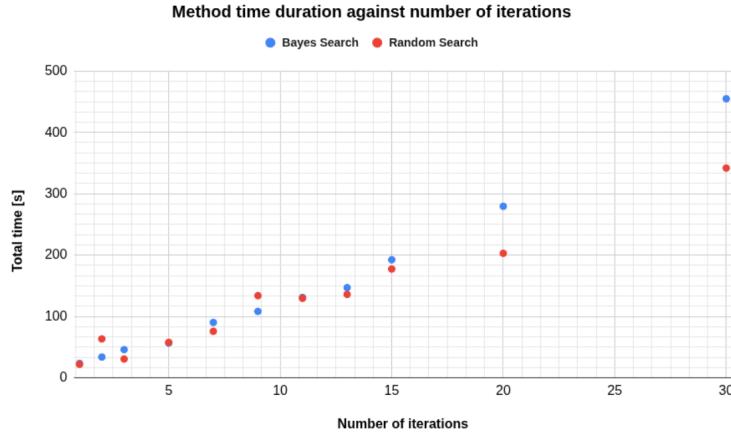
Tabela 6: Porównanie konfiguracji *XGB_default* oraz *Random_covertype* na czterech zbiorach danych.

Na podstawie danych liczbowych z powyższej tabeli 6 można stwierdzić zauważalną poprawę wydajności modelu z konfiguracją *Random_covertype*. Należy jednak zwrócić uwagę, że jest to model dwukrotnie większy w związku z czym uzyskana poprawa może wyciąć z większej "pojemności" modelu. Warto też zwrócić uwagę na

to, że niektóre parametry, jak `learning_rate`, `reg_alpha` lub `subsample` mają zbliżone z konfiguracją domyślną wartości, co przemawia za tezą, że jest ona solidnym i prostym modelem oferowanym przez bibliotekę.

4.4 Porównanie kosztu metod Random Search i Bayes Search

Do przeprowadzenia eksperymentu wykorzystano zbiór *Cover Type* z parametrem kroswalidacji równym 2 wykorzystującym 60% całego zbioru.



Rysunek 2: Czas trwania metod przy rosnącej liczbie iteracji

Uzyskany rezultat odpowiada oczekiwaniom, gdyż optymalizacja metodą Bayesa w każdym kroku wymaga dodatkowych obliczeń w celu zoptymalizowania funkcji akwizycji.

5 Multilayer perceptron

Dla MLP przyjęliśmy siatkę parametrów przedstawioną w 7. Po znalezieniu łącznie 8 zestawów hiperparametrów (dla każdego z czterech zbiorów danych i dwóch sposobów przeszukiwania), przetestowaliśmy każdy zestaw hiperparametrów na każdym ze zbiorów danych. Wyniki porównaliśmy z hiperparametrami defaultowymi `sklearn.neural_network.MLPClassifier`. Nie udało nam się znaleźć takiego zestawu hiperparametrów, który na każdym zbiorze danych uzyskiwał by lepsze wyniki od defaultowych hiperparametrów. Najlepszym zestawem hiperparametrów, jaki udało się nam znaleźć przedstawiliśmy w 8. Osiągnął on lepsze wyniki na 3 z 4 zbiorów. Hiperparametry te pochodzą z Bayesowskiego przeszukiwania przeprowadzonego na zbiorze APSFailure.

Tabela 7: Siatka hiperparametrów dla klasyfikatora Multilayer Perceptron (MLP).

Parametr	Wartości
Rozmiar warstw ukr.	{(16,), (32,), (64,), (128,), (256,)}
Funkcja aktywacji	{relu, tanh}
Współcz. reg. L2 (α)	loguniform(10^{-7} , 10^{-1})
Pocz. szybk. uczenia (η)	loguniform(10^{-4} , 0.5)
Epoki bez poprawy	{10, 20}
Tolerancja optym.	{ 10^{-3} , 3×10^{-3} , 10^{-2} }
Rozmiar paczki	{128, 256, 512}

Tabela 8: Najlepsze znalezione hiperparametry dla klasyfikatora MLP

Parametr	Wartość
Funkcja aktywacji	relu
Współcz. reg. L2 (α)	0.010778
Rozmiar paczki	512
Rozmiar warstw ukr.	(256,)
Pocz. szybk. uczenia (η)	0.003193
Epoki bez poprawy	20
Tolerancja optym.	0.001

6 Wnioski

Ze względu na mnogość metryk porównujących jakość hiperparametrów ciężko jednoznacznie rozstrzygnąć, czy znalezione hiperparametry są istotnie lepsze od parametrów defaultowych. Ponieważ wzięliśmy zróżnicowane zbiory danych, znalezienie hiperparametrów osiągających dobre wyniki dla wszystkich czterech zbiorów danych okazało się problematyczne.

7 Appendix

Hiperparametr	Rozkład	Zakres
n_estimators	randint	[50, 300]
max_depth	randint	[4, 10]
learning_rate	loguniform	[10^{-2} , $3 \cdot 10^{-1}$]
subsample	uniform	[0.5, 1.0)
colsample_bytree	uniform	[0.5, 1.0)
min_child_weight	loguniform	[10^{-1} , 10^1]
reg_lambda	loguniform	[10^{-2} , 10^1]
reg_alpha	loguniform	[10^{-2} , 10^1]
gamma	uniform	[0.0, 2.0)
max_bin	randint	[128, 512]

Tabela 9: Przeszukiwana przestrzeń hiperparametrów

Parameter	Random_car	Random_aps	Random_covertype	Random_jannis	Bayes_car	Bayes_aps	Bayes_covertype	Bayes_jannis	XGB_default
colsample_bytree	0.776	0.547	0.565	0.702	1.000	0.942	1.000	0.500	1.000
gamma	0.140	0.634	0.270	1.496	0.000	0.607	0.245	1.933	0.000
learning_rate	0.063	0.048	0.121	0.043	0.172	0.254	0.194	0.048	0.100
max_bin	128	511	423	390	205	460	128	418	256
max_depth	5	6	8	10	6	4	10	9	3
min_child_weight	2.538	0.281	1.614	0.481	0.100	0.189	0.100	2.873	1.000
n_estimators	292	265	214	165	300	138	300	265	100
reg_alpha	0.029	0.069	0.106	1.391	0.243	0.808	0.010	0.010	0.000
reg_lambda	0.096	6.582	3.705	7.371	0.010	7.252	0.010	5.192	1.000
subsample	0.877	0.880	0.914	0.950	1.000	0.815	0.530	0.881	1.000

Tabela 10: Znalezione konfiguracje hiperparametrów (zaokrąglonych do trzech miejsc po przecinku) oraz domyślna konfiguracja XGB_default.

Literatura

- [1] Breiman, L. (2001). *Random Forests*. Machine Learning, 45(1), 5-32
- [2] Probst, P., Wright, M. N., Boulesteix A. L. *Hyperparameters and Tuning Strategies for Random Forest* (2019).
- [3] Oshiro, T. M., Perez, P. S., Baranauskas, J. A. *How many trees in a random forest?* (2012).