Raport z pracy domowej 2

Łukasz Tomaszewski
Marcel Witas

1 Wstęp

Celem pracy domowej było zbudowanie modeli o jak największej mocy predykcyjnej. Należało dostarczyć dwa modele:

- model zbudowany ręcznie, gdzie sami musieliśmy wybrać model oraz dobrać do niego odpowiednie hiperparametry
- model zbudowany z wykorzystaniem frameworków AutoMLowych, gdzie zadanie wyboru modelu oraz hiperparametrów należało do samego frameworku.

Obydwa modele miały jak najlepiej rozwiązywać zadanie klasyfikacji binarnej na sztucznie wygenerowanym zbiorze *artificial*. Dostarczony zbiór treningowy składał się z 500 zmiennych objaśniających i zawierał 2000 obserwacji. Do zbioru tego dołączona była zmienna celu. Otrzymaliśmy także zbiór testowy, który pozbawiony był zmiennej celu i składał się z 600 obserwacji. Naszym zadaniem było osiągniecie jak największej dokładności predykcji dla obydwu modeli na zbiorze testowym, mierzonej za pomocą miary zrównoważonej dokładności.

2 Model zbudowany ręcznie

2.1 Preprocessing

Dostarczone dane składały się wyłącznie ze zmiennych numerycznych. Nie posiadały one brakujących wartości i każda zmienna miała wariancję różną od 0. Z tego powodu jedynym krokiem preprocessingu było usunięcie silnie skorelowanych kolumn. Usunięte zostały zmienne o korelacji Pearsona większej niż 0.98. Proces ten zmniejszył liczbę kolumn o 10. Dodatkowo przy trenowaniu niedrzewiastych modeli, zmienne objaśniające zostały znormalizowane przy użyciu *StandardScaler*.

Aby być w stanie oszacować dokładność wytrenowanych modeli, musieliśmy podzielić dostarczony zbiór treningowy na zbiór wykorzystywany do uczenia modelu oraz zbiór walidacyjny. Na zbiór walidacyjny składało się 20% obserwacji z dostarczonego zbioru. Dodatkowo proporcja klas w zbiorze walidacyjnym była taka sama jak w zbiorze wyjściowym.

2.2 Trenowanie modeli - BayesianOptimization

Pierwsze eksperymenty zostały przeprowadzone z wykorzystaniem funkcji *BayesianOptimization* z pakietu *bayesian-optimization*.

Jako pierwszy trenowany był XGBoost. Przestrzeń przeszukiwań hiperparametrów dla optymalizacji bayesowskiej przedstawiona została w tabeli 1. Dla tego modelu optymalizacja została uruchomiona 9 razy. W każdym uruchomieniu zwiększana była liczba punktów startowych, zaczynając od 1 i kończąc na 9. Liczba iteracji optymalizacji bayesowskiej w każdym uruchomieniu wynosiła 50. Udało się otrzymać model, którego wynik na zbiorze walidacyjnym wynosił 0.83.

Następnym krokiem było sprawdzenie wpływu poszczególnych zmiennych na predykcję. Zmienne, które nie miały wpływu na predykcję w najlepszym modelu, zostały usunięte. Było to 16 kolumn. Po usunięciu zmiennych powtórzyliśmy eksperyment z XGBoostem. Niestety usunięcie zmiennych nie przyspieszyło uczenia oraz pogorszyło rezultaty. Zdecydowaliśmy się więc powrócić do zbioru z pierwszego eksperymentu.

Następnie trenowane były Catboost oraz Random Forest, na przestrzeni hiperparametrów przedstawionej w tabeli 1. Ponieważ algorytmy te trenowały się dłużej, optymalizacja bayesowska uruchamiana była już tylko 5 razy dla każdego algorytmu. W każdym uruchomieniu liczba iteracji wynosiła 50, a liczba punktów początkowych

zwiększana była co 2, zaczynając od 1 i kończąc na 9. Dla CatBoosta udało osiągnąć się lepsze wyniki niż w przypadku XGBoosta równe 0.8625. Dla Random Forest najlepszy wynik wynosił 0.81.

Tabela 1: Przestrzeń przeszukiwań hiperparametrów

Algorytm	Hiperparametr	Typ	Ograniczenie dolne	Ograniczenie górne
XGBoost	max depth	integer	3	15
	gamma	float	0	1
	min child weight	float	0	5
	subsample	float	0.5	1
	colsample bytree	float	0.5	1
	learning rate	float	0.001	0.3
CatBoost	iterations	integer	100	100
	depth	integer	3	10
	l2 leaf reg	float	1	10
	learning rate	float	0.001	0.3
	subsample	float	0.5	1
Random Forest	n estimators	integer	10	1000
	max depth	integer	1	100
	min samples split	integer	2	10
	max features	float	0.1	0.999

2.3 Trenowanie modeli - ręczne strojenie hiperparametrów

Po zakończonej optymalizacji bayesowskiej postanowiliśmy spróbować znaleźć lepsze hiperparametry metodą prób i błędów. Dla XGBoost oraz Random Forest nie udało nam się poprawić wyników. Strojenie ręczne hiperparametró CatBoosta pozwoliło na osiągnięcie lepszych rezultatów. Osiągneliśmy wynik 0.87 na zbiorze walidacyjnym.

Postanowiliśmy także skorzystać z takich algorytmów jak MLP i SVC. Niestety trenowanie tych modeli nie przyniosło dobrych wyników. Modele te osiągały bardzo dobre rezultaty na zbiorze treningowym, jednak na zbiorze walidacyjnym osiągały wyniki w granicach 0.60. Nie pomagało także zmniejszenie skomplikowania sieci w przypadku MLP czy zmiana jądra w przypadku SVC. Obydwa te algorytmy miały tendencje do przetrenowania i nie radziły sobie ze zbiorem walidacjnym.

2.4 Trenowanie modeli - BayesSearchCV

Ostatnim etapem treningu modeli było uruchomienie funkcji *BayesSearchCV* na CatBoost, który osiągał najlepsze wyniki na zbiorze walidacyjnym. Bazując na wcześniejszych wynikach, poprawiona została przestrzeń przeszukiwań najlepszych hiperparametrów. Przestrzeń, z której brane były wartości przedstawiona została w tabeli 2. Funkcja uruchomiona została na 50 iteracji, w każdej wykonwyana była 3 stopniowa kroswalidacja.

Tabela 2: Przestrzeń przeszukiwań hiperparametrów dla algorytmu CatBoost - BayesSearchCV

Hiperparametr	Typ	Ograniczenie dolne	Ograniczenie górne
iterations	integer	100	1000
depth	integer	6	12
l2 leaf reg	float	6	12
learning rate	float	0.01	0.5
subsample	float	0.8	1

Niestety najlepszy model zwrócony przez funkcję, nie osiągnął lepszego wyniku, niż ten uzyskany przy ręcznym strojeniu hiperparametrów

2.5 Wybór ostatecznego modelu

W tabeli 3 przedstawione zostały najlepsze wyniki na zbiorze walidacyjnym dla poszczególnych modeli.

Tabela 3: Wyniki miary zrównoważonej dokładności dla modeli wybieranych ręcznie

Model	**Zrównoważona dokładność Zbiór walidacyjny**
XGBoost - BayesianOptimization	0.83
XGBoost - usunięte zmienne	0.8025
CatBoost - BayesianOptimization	0.8625
Random Forest - BayesianOptimization	0.81
XGBoost - Ręczne strojenie	0.83
CatBoost - Ręczne strojenie	**0.87**
Random Forest - Ręczne strojenie	0.81
MLP - Ręczne strojenie	0.58
SVC - Ręczne strojenie	0.59
CatBoost - BayesSearchCV	0.8625

Do wykonania predykcji na zbiorze testowym wybrany został CatBoost z ręcznie dostrojonymi hiperparametrami.

3 Model zbudowany przy użyciu AutoML

3.1 Wybór frameworku

Eksperymenty zostały przeprowadzone przy użyciu dwóch frameworków.

Pierwszy z nich to *MLJAR* [2], o którym mieliśmy okazję usłyszeć na jednym z wykładów. Został wybrany ze względu na łatwą instalację, a także dość łatwą możliwość dostosowania go do oczekiwań. Drugi pakiet, który wykorzystaliśmy, to *AutoSklearn 1* [1]. Jego mocną stroną są wbudowane metody preprocessingu, który wydawał się ważny przy zbiorze danych zawierającym bardzo dużo zmiennych objaśniających.

3.2 Eksperymenty - MLJAR

Na początku podjęliśmy próby użycia funkcji `AutoML` z pakietu MLJAR z domyślnymi parametrami. Już wtedy udało się uzyskać wartość metryki balanced accuracy na zbiorze walidacyjnym około 0.8, zatem całkiem obiecujący wynik. Niestety w tym frameworku nie ma możliwości wybrania jako optymalizowanej metryki balanced accuracy. Zdecydowaliśmy się zatem, aby podać accuracy, która przy równej liczbie klas (co występowało w naszym zbiorze danych) jest tożsama z balanced accuracy.

Następnie zaczęliśmy zmieniać pozostałe parametry funckji. Argument `mode` ustawiliśmy na wartość 'Compete', która według autorów pakietu pozwala osiągnąć najlepsze wyniki. Sprawdziliśmy, że najlepsze wyniki osiągały modele oparte o drzewa jak ExtraTress, Xgboost, Catboost. Po podaniu odpowiedniego argumenty, algorytm używał tylko tych modeli.

Ze względu na liczbę kolumn w zbiorze danych, ustawiliśmy argument `features_selection` na True. Okazał się on jednak problematyczny, ponieważ w pierwszych próbach ten krok był pomijany, wyświetlając komunikat o zbyt niskim limicie czasu. Po zwiększeniu limitu, funkcja dawała więcej czasu na trenowanie algorytmów, i znów pomijała krok związany z selekcją zmiennych, nawet gdy limit wynosił 20 minut.

Wykorzystaliśmy zatem liczbę informacji genberowanych przez framework, i z algorytmu, który okazał się najlepszy w jednej z iteracji, wyciągnęliśmy feature importance, który był niezerowy dla 87 kolumn. Zatem ograniczyliśmy zbiór danych i na nim MLJAR osiągnął najlepsze wyniki. Znów korzystaliśmy tylko z modeli ExtraTress, Xgboost i Catboost. Wyniki uzyskane przez pakiet MLJAR można zobaczyć w tabeli 4.

3.3 Eksperymenty - AutoSklearn 1

Następnym przetestowanym frameworkiem był auto-sklearn w wersji pierwszej. W nim jest możliwość wyboru balanced accuracy jako optymalizowanej metryki. Pierwsze eksperymenty wyglądały również bardzo obiecująco, bo w krótkim czasie udało się uzyskać wartości około 0.84 na zbiorze walidacyjnym, czyli lepsze niż niektóre modele MLJAR.

Następnie przeprowadzono trening ustalając limit czasu na 10 minut, i `resampling_strategy` na 'holdout'. Już wtedy udało się uzyskać nieznacznie lepszy wynik, niż dla najlepszego modelu MLJAR.

Potem, korzystając z doświadczenia przy wcześniejszym treningu, ustalono przestrzeń algorytmu na Extra Trees i Random Forest. Po 10 minutach uzyskaliśmy model, który osiągnął 0.89 na zbiorze walidacyjnym. W skład ostatecznego ensembla wchodziły modele, przy preprocessingu używające metod: Fast Independent Component Analysis, Select Percentile Classification, Extra Trees Classifier Preprocessing i Random Trees Embedding; a jako algorytmu trzy z nich to Extra Trees, a jeden Random Forest.

3.4 Wybór ostatecznego modelu

Ponieważ Autosklearn uzyskał najlepsze wyniki metryki balanced accuracy na zbiorze walidacyjnym, co przedstawia tabela 4, to on został wybrany do użycia predykcji na zbiorze testowym.

Model	**Czas treningu (w minutach)**	**Zrównoważona dokładność Zbiór treningowy**	**Zrównoważona dokładność Zbiór walidacyjny**
MLJAR	10	0.9463	0.845
MLJAR	20	0.9544	0.83
MLJAR: kolumny wybrane na podstawie Feature Importance	12	**0.9819**	0.86
Autosklearn: domyślne parametry	10	0.9638	0.8625
Autosklearn: Extra Trees i Random Forest	10	0.9588	**0.89**

Tabela 4: Wyniki metryki balanced accuracy (zrównoważona dokładność) dla testowanych frameworków AutoML z różnymi parametrami.

4 Podsumowanie

Celem projektu było przygotowanie dwóch wariantów metod klasyfikacji, które pozwolą zbudować model o jak największej mocy predykcyjnej dla sztucznie wygenerowanego zbioru *artificial*. Modele przygotowane przez nas osiągnęły zadowalające wyniki metryki zrównoważonej dokładności (0.87 i 0.89) na wydzielonym przez nas zbiorze walidacyjnym. Model zbudowany przy użyciu frameworku AutoMLowego osiągnął nieznacznie lepszy wynik metryki, co pokazuje ogromny potencjał takich rozwiązań.

Literatura

[1] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Advances in Neural Information Processing Systems 28 (2015)*, pages 2962–2970, 2015.

[2] Aleksandra Płońska and Piotr Płoński. Mljar: State-of-the-art automated machine learning framework for tabular data. version 0.10.3, 2021.