automl-hw2

Szymon Gut
Maciej Orsłowski

December 2023

1 Wprowadzenie

Celem zadania było wytrenowanie modelu, o jak najlepszej predykcji dla udostępnionych danych. Projekt zakładał zwrócenie predykcji dwóch modeli: pierwszy to model wytrenowany samodzielnie, drugi natomiast to model zwrócony przez jeden z pakietów AutoML'owych.

2 Dane

Zbiór danych został wygenerowany sztucznie i podzielony na dane treningowe oraz testowe. Dostarczony do nas został jedynie pełny zbiór treningowy oraz dane testowe bez etykiet, na których należało dokonać finalnej predykcji.

W zbiorze treningowym znajdowało się 500 zmiennych predykcyjnych, wszystkie numeryczne i składał się on z 2000 obserwacji. Zmienna celu składała się z dwóch kategorii: -1 oraz 1, każda z nich miała po 1000 wystąpień. W zbiorze testowym natomiast znajdowało się 600 obserwacji.

Od razu po wczytaniu danych, zmienna celu została przez nas przekodowana: wartości -1 zostały zamienione na 0. Zostało to spowodowane przede wszystkim faktem, że niektore modele (jak np. XGBoost) nie działały dla oryginalnego kodowania.

3 Preprocessing

W fazie preprocessingu, na samym wstępie warto zadbać o podział danych, które mieliśmy otagowane (*artificial_train*) na zbiór treningowy oraz walidacyjny (w stosunku 80:20). Jest to niezbędny krok do zbadania wydajności naszego modelu, gdyż tak jak zostało wspomniane, jego performence na zbiorze testowym będzie można wyznaczyć dopiero po opublikowaniu prawidłowcy etykiet do tego zbioru.

Dalsza część preprocessingu wykorzystywana była jedynie przy podejściu manualnym. W pierwszej kolejności ze zbioru usunięte zostało 25% najmniej skorelowanych ze zmienną celu kolumn. Wykorzystanym tu rodzajem korelacji

była korelacja Spearmana z uwagi na możliwość przez nią wykrycia zależności nieliniowych. Była ona wyliczana na zbiorze treningowym.

W dalszej części stworzono Pipeline, który przeprowadzał preprocessing na danych wejściowych wykonując odpowiednio:

1. Imputacje braków danych (zastępując je średnią), w tym celu wykorzystany został `SimpleImputer` z biblioteki *scikit-learn*;
2. PCA, wybierając liczbę komponentów równą 170, z uwagi, iż w ten sposób mieliśmy zapewnioną wyjaśnialność wariancji na poziomie 90% (zobacz wykres 1), redukując przy tym wymiarowość danych mniej więcej o połowę;
3. Skalowanie danych: w tym celu został wykorzystany `MinMaxScaler` z biblioteki *scikit-learn*.

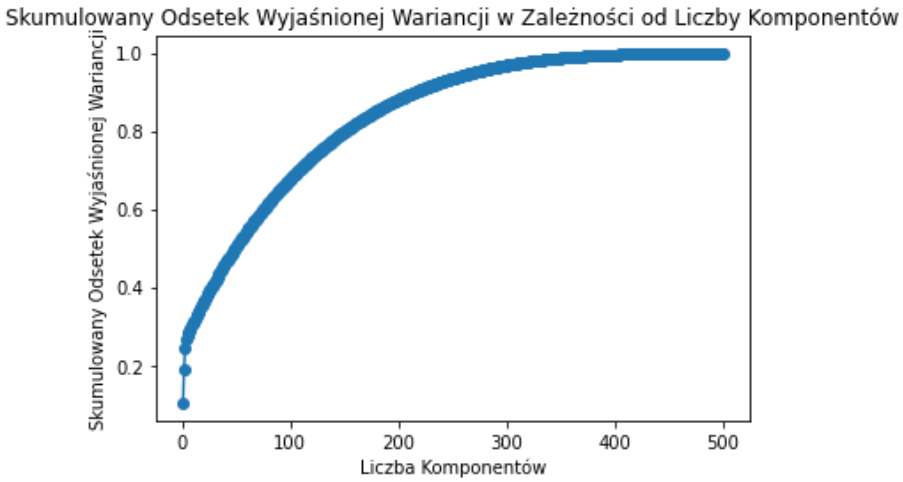

Figure 1: Procent wyjaśnianej wariancji w zależności od liczby komponentów

4 Modele

Przechodząc do własnoręcznie trenowanych modeli, podczas ich tuningu, została zastosowana optymalizacja Bayesowska, z uwagi na lepsze wyniki tej metody uzyskane podczas realizacji pracy domowej HW1, w której badaliśmy podatność różnych modeli na tunowalność oraz wpływ metody samplingu (Random Search oraz Bayes Search).

Jakość modeli była mierzona metryką balanced accuracy, wyliczaną za pomocą wzoru:

$$balanced\_accuracy = \frac{1}{2}(\frac{TP}{P} + \frac{TN}{N})$$

Wykorzystane modele to:

- **XGBoost** - algorytm drzewiasty wykorzystujący metode wzmacniania gradientowego
- **Random Forest** - drzewiasty algorytm zahaczający ideę baggingu
- **sieć neuronowa MLP** z funkcjami aktywacji ReLU, entropią krzyżową jako funkcją straty, optimizerem Adam oraz funkcją softmax w warstwie wyjściowej, zwracającej wektor prawdopodobieństwa przynależności do każdej z klas

Przestrzeń parametrów przy optymalizacji hiperparametrów dla modelu XGBoost przedstawiona jest w tabeli 1, zaś dla Random Forest w tabeli 2.

Table 1: Przestrzeń hiperparametrów dla algorytmu XGBoost

Parametr	**Przestrzeń**
n_estimators	$1, \ldots, 5000$
eta	$[2^{-10}, 1.0]$
subsample	$0.1, 1.0$
max_depth	$1, \ldots, 15$
min_child_weight	$[1, 2^7]$
colsample_bytree	$[0.0, 1.0]$
alpha	$[2e-10, 2e10]$

Table 2: Przestrzeń hiperparametrów dla algorytmu RandomForest

Parametr	**Przestrzeń**
n_estimators	$1, \ldots, 2000$
max_depth	$3, \ldots, 500$
max_features	$[10^{-6}, 1]$
max_samples	$[10^{-6}, 1]$
min_samples_leaf	$[0.1, 0.5]$

Pakiety AutoML, które zostały użyte do znalezienia modelu o najwyższej skuteczności przy predykcji to *AutoGluon* oraz *MLJAR*.

5 Wyniki

Skuteczność poszczególnych modeli została sprawdzona w pierwszej kolejności na zbiorze walidacyjnym utworzonym przez nas samych, zaś na koniec również na próbce zbioru testowego w aplikacji udostępnionej przez prowadzących. Z uwagi jednak na mały rozmiar udostępnionych danych testowych (5% zbioru, co daje 30 obserwacji), wynik uzyskany na tym zbiorze traktowany był nie jako

Table 3: Wyniki modeli

Podejście	Metoda	Score walidacyjny	Score testowy
Manualne	**XGBoost**	**0.7704**	**0.9000**
	Random Forest	0.6372	0.8333
	MLP	0.6832	-
AutoML	**MLJAR**	**0.8712**	**0.8333**
	AutoGluon	0.8709	0.7667

główne kryterium wyboru modeli, tylko raczej jako czynnik rozstrzygający w sytuacji, gdy modele osiągały podobny wynik na danych walidacyjnych.

Pełne wyniki zostały przedstawione w tabeli 3. Score był liczony jako balanced accuracy.

Wśród modeli wykorzystanych w podejściu manualnym zdecydowanie wyróżnia się XGBoost, który zdobył najlepsze wyniki zarówno na zbiorze walidacyjnym, jak i próbce z testowego. Wśród podejścia manualnego należy jednak wyjaśnić sytuację z MLP, dla którego nie został wyliczony wynik na testowym. Było tak dlatego, że w momencie jego wytrenowania nie było jeszcze dostępu do próbki zbioru testowego, a jako, że wyniki sieci nie były zadowalające, jej parametry nie zostały przez nas zapisane na później. Stąd, gdy dostęp ten został w końcu przyznany, sprawdzenie wyniku na nim wymagałoby ponownego wytrenowania sieci, co z uwagi na długi czas trwania treningu i słabe wyniki przez nią osiągane nie miało większego sensu.

Jeśli chodzi o podejście AutoML, to tutaj oba frameworki osiągnęły niemal identyczny wynik na zbiorze walidacyjnym. Nie mogąc zdecydować się który z nich wybrać do stworzenia finalnej predykcji, zdecydowaliśmy się porównać ich wyniki na próbce zbioru testowego, które okazały się być lepsze dla frameworka *MLJAR*.