AutoML - FLAML

Grupa KTR

1 Wstęp (zarys)

Cytować będziemy:

- https://microsoft.github.io/FLAML/docs/reference/model – wszystko o modelu
- https://new.openml.org/search?type=benchmarksort=tasks_includedstudy_type=taskid=218 - zbiory danych do wstępnej ewaluacji funkcji

2 Opis frameworku FLAML

FLAML (Fast and Lightweight AutoML) jest biblioteką języka Python, wydaną przez Microsoft na licencji Open Source, służącą do automatycznego uczenia maszynowego (AutoML). Według twórców FLAML jest lekkim pakietem pozwalającym na znalezienie odpowiedniego modelu uczenia maszynowego do danego problemu w sposób automatyczny, efektywny i ekonomiczny. W założeniu więc ma zwalniać użytkownika z konieczności wyboru optymalnego modelu uczenia maszynowego oraz jego hiperparametrów. Framework ten wspiera zarówno tradycyjne metody uczenia maszynowego - pozwala szybko znaleźć dobre jakościowo modele używając niewielkich zasobów obliczeniowych dla często spotykanych zadań, takich jak klasyfikacja czy regresja - jak i uczenie głębokie (deep learning).
Główną klasą pakietu jest `flaml.AutoML`. Jej konstruktor przyjmuje ustawienia działania - budżet czasowy, wykorzystywana metryka, zadanie do wykonania (klasyfikacja, regresja etc.), lista modeli i inne. Najważniejszą metodą, podobnie jak w pakiecie scikit-learn jest metoda `fit()` przyjmująca zbiór danych objaśniających X oraz kolumnę objaśnianą y (możemy w tej metodzie także nadpisać ustawienia). Wykonując ją trenujemy dane na modelach odpowiednich dla danego typu problemu oraz dokonujemy dla nich optymalizacji hiperparametrów. Dzięki odpowiednim atrybutom `best_estimator` oraz `best_config` jesteśmy w stanie pozyskać najlepszy model oraz konfigurację jego hiperparametrów znalezionych przez FLAML.
Oprócz `fit()` można wywołać też metody `predict()` oraz `predict_proba()`, dzięki czemu zachowana jest kompatybilność z pakietem scikit-learn.
Atutem FLAML jest także możliwość personalizacji działania oprogramowania - pakiet może obsłużyć funkcję zdefiniowaną przez użytkownika (najprościej do

tego użyć funkcji kompatybilnych z pakietem scikit-learn, a więc zawierających metody `fit()` oraz `predict()` oraz korzystać z zewnętrznej metryki.
Moduł `tune` biblioteki pozwala na ekonomiczne dobieranie odpowiednich parametrów do danego modelu. Jest używane wewnętrznie przez `flaml.AutoML`, lecz może być także bezpośrednio wykorzystany przez użytkownika, spełniając jeden z warunków:

- zadanie użytkownika nie jest jednym z wbudowanych zadań w ramach `flaml.AutoML`
- dane wejściowe użytkownika nie mogą być przedstawione w postaci kolumn ze zmiennymi objaśniającymi oraz kolumny objaśnianej
- użytkownik chce dokonać tuningu funkcji niekwalifikującej się jako procedura uczenia maszynowego

Dużą wadą tej paczki jest to, że dokumentacja do niej pozostawia wiele pytań. Na przykład preprocessing, który model wykonuje nie być nigdzie opisany, jedynie była w SDK wspomniana klasa DataTransformer, przez to wykonaliśmy własny preprocessing.

3 Czynności konieczne do podjęcia aby pakiet prawidłowo działał

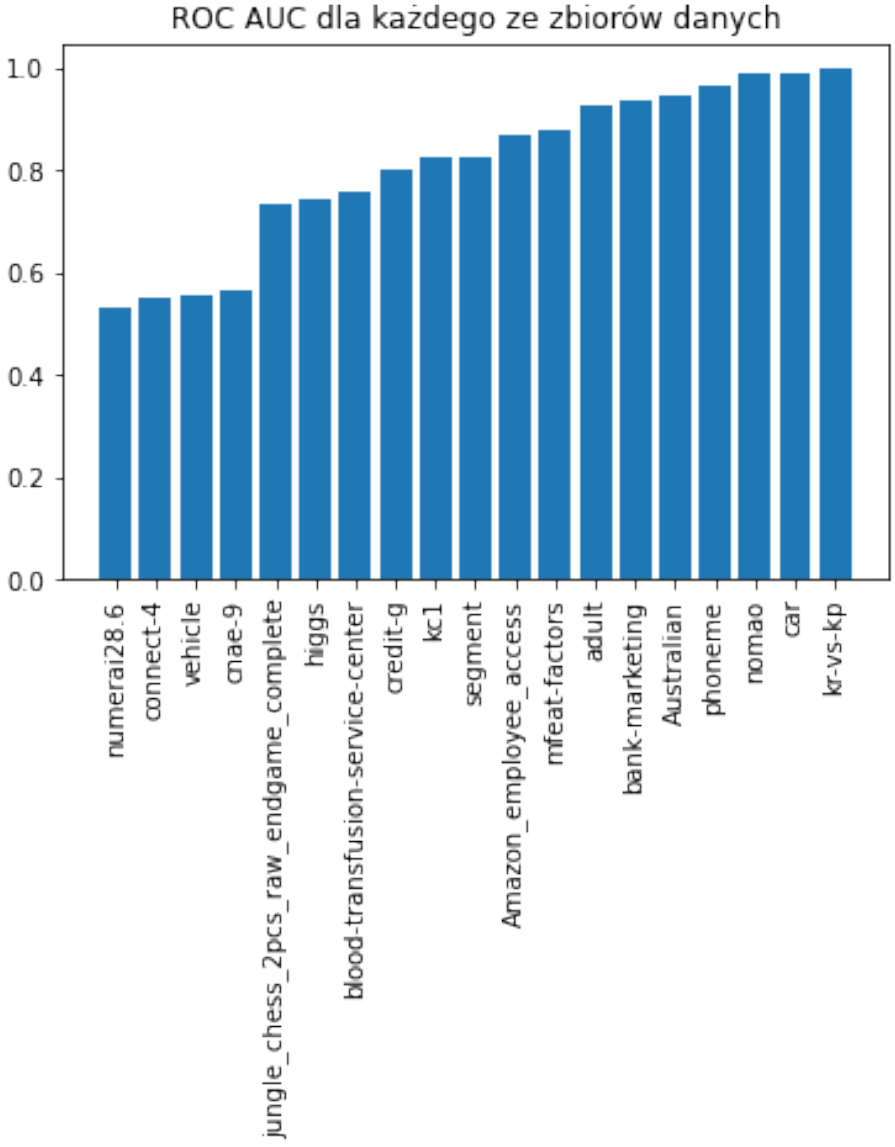
Ze względu na kiepską dokumentację, myśleliśmy, że w pakiecie nie ma preprocessingu. W związku z tym aby uniknąć niezdefiniowanych zachowań musimy wykonać jakiś preprocessing dla danych, aby użytkownik nie musiał się tym zajmować. Preprocessing na który się zdecydowaliśmy to:

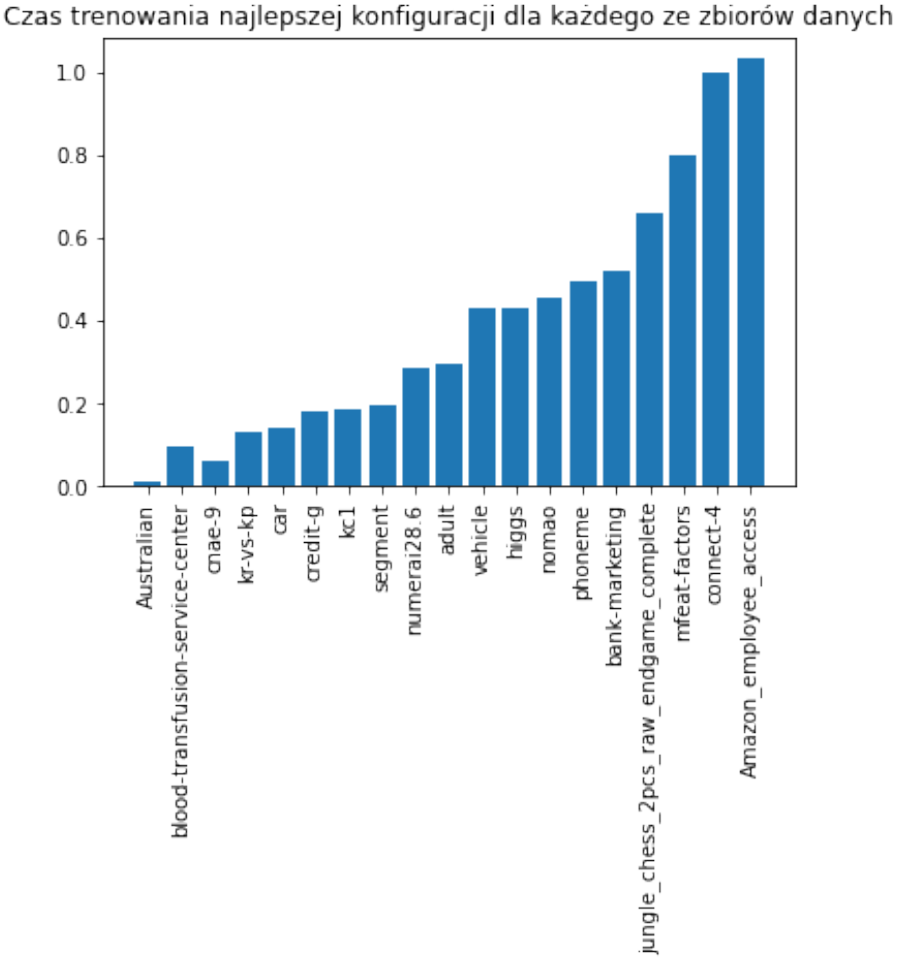
- usunięcie kolumn będących id
- usunięcie kolumn o zerowej wariancji
- imputacja danych - metodą mody
- zastąpienie ewentualnych outlierów wartością graniczną
- normalizacja rozkładu za pomocą QuantileTransformer
- przeskalowanie danych za pomocą StandardScaler
- encoding zmiennych kategorycznych za pomocą WoEEncoder

Wybraliśmy taki preprocessing, aby nie tylko zagwarantować brak niezdefiniowanych zachowań w przypadku klasyfikacji binarnej, ale także aby uniknąć przetrenowania i otrzymywać lepsze modele. Okazuje się jednak, że model wykonuje preprocessing potrzebny aby działał. Podobnie jak my usuwa kolumny o zerowej wariancji i imputuje braki danych albo nową kategorią w przypadku zmiennych kategorycznych, albo medianą w przypadku zmiennych numerycznych oraz rozbija datę na jej składowe.

4 Wyniki ewaluacji

W ramach wstępnej ewaluacji rozwiązania dokonaliśmy sprawdzenia osiągów utworzonej funkcji na dwudziestu zbiorach danych z OpenML, zawartych w zadaniu dostępnym pod linkiem (nie są to wszystkie zbiory zawarte w tym zadaniu). 13 spośród nich jest klasyfikacją binarną. Na każdym ze zbiorów wykonaliśmy utworzoną przez nas metodę `fit()` z utworzonej przez nas klasy `OurFlaml` (aby do tego doszło musieliśmy pobrać każdy ze zbiorów i podzielić go na kolumny objaśniające i kolumnę objaśnianą). Przypisaliśmy do odpowiednich list najlepszy wynik uzyskany w ramach funkcji dla danego zbioru, czas wytrenowania najlepszej konfiguracji oraz nazwę zbioru. Stosowaną do tego metryką był ROC AUC. Uzyskane wyniki zwizualizowaliśmy:


ROC AUC dla każdego ze zbiorów danych
1.0
0.8
0.6
0.4
0.2
0.0
numerai28.6
connect-4
vehicle
cnae-9
jungle_chess_2pcs_raw_endgame_complete
higgs
blood-transfusion-service-center
credit-g
kc1
segment
Amazon_employee_access
mfeat-factors
adult
bank-marketing
Australian
phoneme
nomao
car
kr-vs-kp


Czas trenowania najlepszej konfiguracji dla każdego ze zbiorów danych
1.0
0.8
0.6
0.4
0.2
0.0
Australian
blood-transfusion-service-center
cnae-9
kr-vs-kp
car
credit-g
kc1
segment
numerai28.6
adult
vehicle
higgs
nomao
phoneme
bank-marketing
jungle_chess_2pcs_raw_endgame_complete
mfeat-factors
connect-4
Amazon_employee_access