Code review

Grzegorz Zbrzeżny

May 2022

1 Wstęp

Moim zadaniem była weryfikacja kodu napisanego przez grupę Gakubu, zajmującą się pakietem AutoGluon. Kod ten rozdzielono na dwa pliki *utils.py* i *notebook.ipynb*. W tym pierwszym znajdowały się funkcje potrzebne do pobrania plików w formacie .ARF oraz do przeprowadzenia K Foldowej cross - validacji, natomiast drugi zawierał pętle potrzebne do wywołania kodu z pliku pierwszego dla wszystkich wymaganych zbiorów danych.

2 Ocena kodu

2.1 Realizacja celów

Kod zdecydowanie osiąga postawiony cel, ponieważ na podstawie podanych mu linków URL do zbiorów danych z OpenML'a był w stanie przetworzyć te dane na pandasowe ramki danych, przeprowadzić odpowiedni preprocessing oraz cross - validację na zadanej ilości foldów. Nie doszukałem się w nim żadnych błędów logicznych, kod krok po kroku realizuje zadane cele w zwięzły i zoptymalizowany sposób bez niepotrzebnych udziwnień. W prezentacji opisującej pakiet AutoGluon grupa Gakubu nie zawarła dokładnych wytycznych dotyczących potrzebnego przygotowania danych, aby pakiet mógł na nich zadziałać, skupili się oni na opisie możliwości pakietu. Jednakże trzeba stwierdzić, że zaimplementowany preprocessing najprawdopodobniej jest wystarczający, ponieważ zadziałał on zarówno na zbiorach z OpenML'a jak i na pobranym przeze mnie z Kaggle'a secie.

2.2 Zgodność ze standardami PEP 8

Aby zbadać zgodność ze standardami PEP 8 użyłem pakietu Pylint. Po użyciu go na pliku zawierającym funkcję zespołu Gakubu okazało się, że jest w nim trochę niezgodności z tym standardem. Były to:

- zbyt duzo znaków w jednej linijce kodu
- brak docstringa dla całego modułu
- funkcja przyjmuje zbyt dużo argumentów
- nazwy niektórych zmiennych nie odpowiadającą konwencji *snake case*
- zła kolejność importów

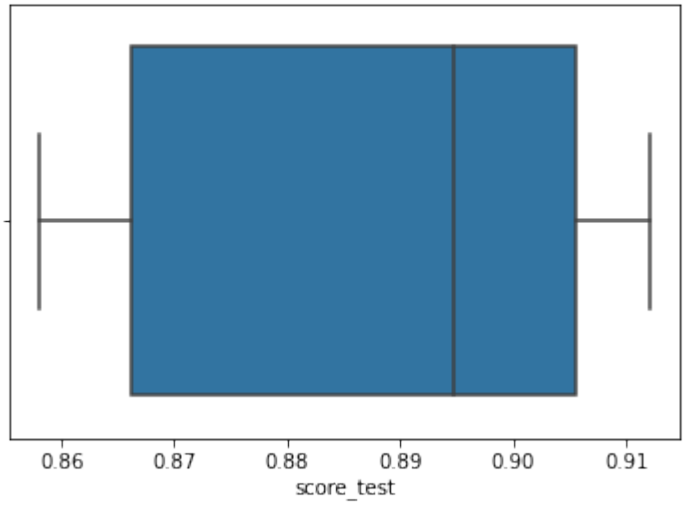
Jednakże poza tymi drobnymi problemami kod był zgodny ze standardami.

2.3 Ogólna ocena

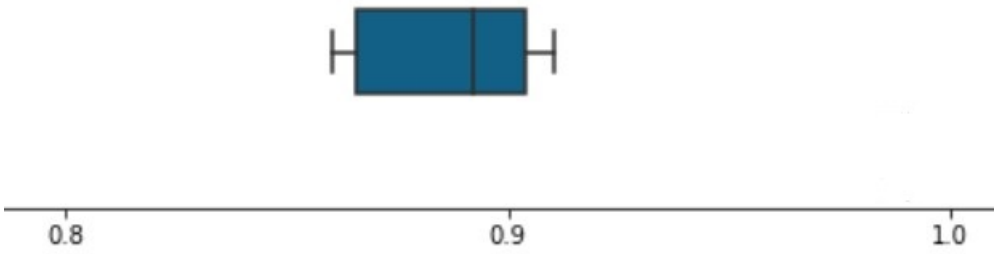
W kodzie nie ma elementów, które zdecydowanie nadawałyby się do poprawy, bądź optymalizacji, wszelkie operacje wykonywane są w sposób efektywny, a sam kod nie zawiera linijek, które można by zapisać o wiele krócej. Dokumentacja w postaci docstringów jest wystarczająca, po jej przeczytaniu wiadomo co dana funkcja robi, dużo o działaniu funkcji można wywnioskować z samych ich nazw. Komentarze również są wystarczające.

3 Reprodukowalność wyników

Aby ocenić reprodukowalność wyników puściłem oceniany kod na zbiorze *jasmine*, był to jeden z pobranych z OpenML'a zbiorów, na których mieliśmy sprawdzić nasz pakiet do Kamienia Milowego 2. Wyniki najlepszych modeli w 10 foldach przedstawiłem za pomocą boxplota:



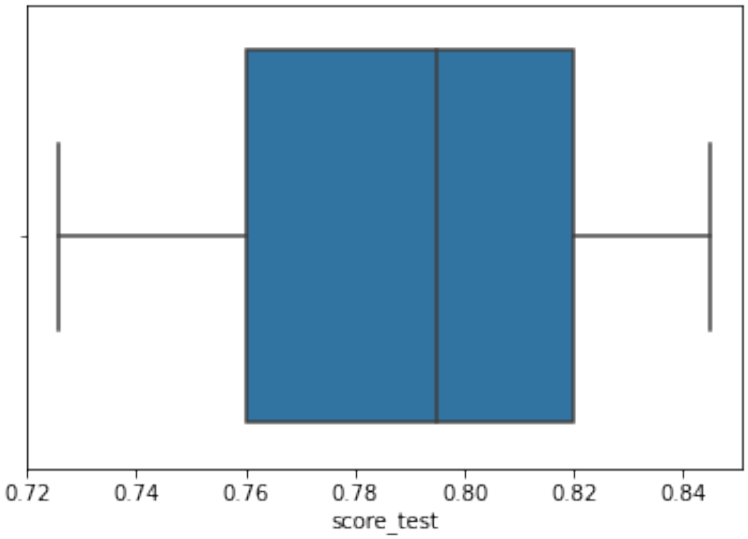
Natomiast wyniki grupy Gakubu dla tego zbioru wyglądały następująco:



Wyniki osiągnięte przeze mnie są bardzo zbliżone do tych uzyskanych przez zespół tworzący funkcję.

4 Działanie na innych zbiorach

Aby sprawdzić jak funkcja poradzi sobie z innymi danymi pobrałem zbiór *Student Alcohol Consumption* z portalu Kaggle. Za pomocą tego datasetu chciałem przewidzieć do której szkoły uczęszczają studenci, żeby problem dotyczył klasyfikacji binarnej (w zbiorze przedstawiono dane studentów z dwóch szkół). Tym razem włączyłem funkcję na trzech foldach, ponieważ przy dziesięciu otrzymywałem błąd *[ErrNo 28] No space left on device*. Wyniki zebrałem w postaci boxplota:



Funkcja zadziałała na nowych danych dając całkiem niezłe wyniki. Jedynym problemem, który napotkałem był błąd w przypadku, gdy etykieta była dana typem Object (w moim przypadku String). Preprocessing zaimplementowany przez grupę Gakubu zamieniał ją wtedy na kolumnę wartości *NaN*. Dopiero po ręcznej zamianie wartości w etykiecie na 0 i 1 udało mi się puścić funkcję dalej.