Recenzja kodu: Moja grupa, Auto-sklearn 2.0

Kacper Kurowski

4 maja 2022

Spis treści

1 Uwagi wstępne

Przed przejściem do faktycznej recenzji kodu przedstawię kilka uwag wstępnych:

- Kod był kompilowany w najnowszej (na chwilę obecną — dzień 4 maja 2022) wersji pakietu auto-sklearn. Pobrane zostały wówczas najnowsze wersje pakietów zależnych. (Wyjątkiem był pakiet scipy, który został użyty w wersji 1.7.3, gdyż najnowsza wersja pakietu, 1.8.0, wprowadzona 2 maja 2022 powodowała błąd)
- Autorzy kodu wykorzystywali w swoim notatniku pliki w formacie .PKL zawierające zbiór danych, a także indeksy wykorzystywane w poszczególnych foldach. Niestety nie byłem pewien, gdzie mogę zdobyć wspomniane pliki, więc zdecydowałem się dostosować zbiory danych w sposób, jaki uważałem, że powinienem uzyskać po opuszczeniu funkcji *load_data*. Nie mogłem zatem sprawdzić funkcjonalności wspomnianej funkcji, ale znajdowała się ona poza plikiem .PY.

Rysunek 1: Uruchomienie kros-walidacji

- Jako że nie mogłem być pewien, jakie informacje przechowywały wspomniane pliki .PKL, konieczne było niewielkie dostosowane danych, by uruchomić funkcję *cross_valid* — polegało ono jednak tylko na zmienieniu wartości zmiennej celu na numeryczne.
- Brak plików .PKL oznaczał również konieczność dokonania podziału na foldy poza funkcją. W tym celu użyta została funkcja *KFold* od *sklearn.model_selection*. Ze względu na sposób odzyskiwania foldów w załączonym pliku .PY utworzona również została klasa przechowująca dwa pola: *.train* oraz *.test* zawierające listy indeksów przypadających na poszczególne części folda.
- 3 maja blisko godizny 15:00 do pliku .PY została dodana funkcja dokonująca podziału na foldy oraz dwa docstringi. Niektóre z uwag, a także zdjęć dotyczą stanu przed rzeczoną zmianą.

2 Recenzja kodu

2.1 Czy kod osiąga cel, który postawiono?

Tak, choć uruchomienie kros-walidacji wymaga posiadania uprzednio przygotowanego podziału na foldy, który zgodnie z notatnikiem znajdował się w pliku .PKL. (3 maja została dodana funkcja, która robi podział na foldy automatycznie)

Po odpowiednim przygotowaniu zmiennych predykujących, zmiennej celu oraz indeksów foldów (proces ten dokładniej omówiłem w Uwagach wstępnych), uruchomienie kros-walidacji jest już proste (patrz Rysunek 1). W rezultacie otrzymujemy dwie listy wyników, gdyż policzone zostały dwie metryki: *accuracy* oraz *auc* (patrz Rysunek 2).

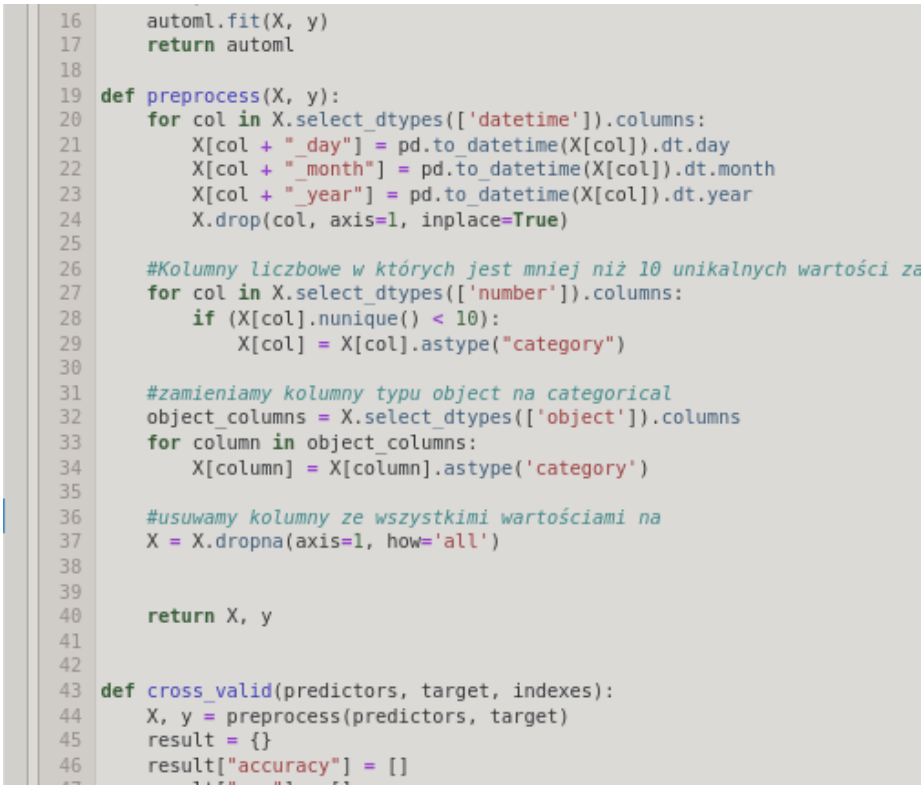
2.2 Czy w kodzie są jakieś oczywiste błędy logiczne?

Nie znalazłem żadnych.

```
100]: score

100]: {'accuracy': [0.8350051177072672,
  0.8235414534288639,
  0.8284193284193284,
  0.828009828009828,
  0.8251433251433251,
  0.8194103194103194,
  0.8245290745290745,
  0.833947583947584,
  0.8247338247338247,
  0.8341523341523341],
 'auc': [0.9237751684054251,
  0.9231255766883948,
  0.9203614460028985,
  0.9250675593862141,
  0.9211766228373441,
  0.919053989044045,
  0.9245139665575853,
  0.9251327861105113,
  0.9285447712184444,
  0.9255453744509282]}
```

Rysunek 2: Wyniki uzyskane w kros-walidacji



```
    automl.fit(X, y)
    return automl

def preprocess(X, y):
    for col in X.select_dtypes(['datetime']).columns:
        X[col + "_day"] = pd.to_datetime(X[col]).dt.day
        X[col + "_month"] = pd.to_datetime(X[col]).dt.month
        X[col + "_year"] = pd.to_datetime(X[col]).dt.year
        X.drop(col, axis=1, inplace=True)

    #Kolumny liczbowe w których jest mniej niż 10 unikalnych wartości za
    for col in X.select_dtypes(['number']).columns:
        if (X[col].nunique() < 10):
            X[col] = X[col].astype("category")

    #zamieniamy kolumny typu object na categorical
    object_columns = X.select_dtypes(['object']).columns
    for column in object_columns:
        X[column] = X[column].astype('category')

    #usuwamy kolumny ze wszystkimi wartościami na
    X = X.dropna(axis=1, how='all')


    return X, y


def cross_valid(predictors, target, indexes):
    X, y = preprocess(predictors, target)
    result = {}
    result["accuracy"] = []
```

Rysunek 3: Fragment kodu. Odstęp między pierwszą funkcją a drugą to jeden pusty wiersz, odstęp między drugą, a trzecią, to dwa puste wiersze

2.3 Czy patrząc na wymagania zawarte podczas prezentacji są one w pełni zaimplementowane?

Niektóre z wymagań (lub wyjaśnień wymagań) mogły być zawarte w zdjęciach, które zdają się nie być dostępne w udostępnionym notatniku (w którym znajduje się prezentacja). Potwierdza się natomiast możliwość policzenia metryk: *accuracy* oraz *auc*.

2.4 Czy kod jest zgodny z istniejącymi wytycznymi stylistycznymi? (czy kod jest zgodny z PEP 8)

- konwencja nazewnicza jest zgodna z PEP 8 (przy założeniu, że stosujemy raczej typową konwencję oznaczania zmiennych predykujących przez X, a zmienną celu symbolem y.
- Konwencja nowych linii dla funkcji poza klasą nie jest zgodna z PEP 8 (patrz Rysunek 3).

```
43 def cross_valid(predictors, target, indexes):
44     X, y = preprocess(predictors, target)
45     result = {}
46     result["accuracy"] = []
47     result["auc"] = []
48 
49     for index in indexes:
50         X_train = X.iloc[index.train]
51         y_train = y.iloc[index.train]
52         model = fit_automl(X_train, y_train)
53 
54         X_test = X.iloc[index.test]
55         y_test = y.iloc[index.test]
56 
57         y_pred = model.predict(X_test)
58         y_proba = model.predict_proba(X_test)[:,1]
59         result["accuracy"].append(accuracy_score(y_test, y_pred))
60         result["auc"].append(roc_auc_score(y_test, y_proba))
61     return result
```

Rysunek 4: Konwencja stosowania białych znaków wokół znaku = jest zachowana

Występuje jednak poprawne oddzielanie logicznych fragmentów poszczególnych funkcji przez puste wiersz (oprócz polecenia *return*, które ma nieregularne odstępy od ciała funkcji),

- Liczba znaków w wierszu jest zachowana (oprócz jednego komentarza, który ma o 13 znaków za wiele). Indentacja jest prawidłowa,
- Komentarze są używane prawidłowo, brakuje jednak docstringów. (Przy zmianie 3 maja (około godziny 15:00), dwie funkcje posiadają już docstringi)
- Białe znaki (poza przypisywaniem argumentów domyślnych funkcji — patrz Rysunek 5) są stosowane prawidłowo — patrz Rysunek 4

Odstępy między argumentami funkcji po przecinku sa prawidłowe. Można również rozważyć pozbycie się nawiasów wokół predykatu po (jedynym) poleceniu *if* w kodzie.

2.5 Czy są jakieś obszary, w których kod mógłby zostać poprawiony? (skrócić, przyspieszyć, itp.)

Nie znalazłem żadnych, które mogłyby być istotne.

```
def fit_automl(X:pd.DataFrame, y:pd.DataFrame):
    #daty na wejsciu musza byc typu datetime
    automl = AutoSklearn2Classifier(
        time_left_for_this_task = 60,
        per_run_time_limit = 27, # mniej niż połowa time_left_for_this_task
        disable_evaluator_output=False,
        n_jobs = -1,
        memory_limit = None,
        metric=roc_auc
        )
    automl.fit(X, y)
    return automl

```

Rysunek 5: Konwencja stosowania białych znaków wokół znaku = dla podawania argumentów funkcji nie jest zachowana. Inna funkcja dodana 3 maja zachowuje jednak tę konwencję

2.6 Czy dokumentacja i komentarze są wystarczające?

Cenna anotacja w postaci typu *pd.DataFrame*, która jest obecna w definicji funkcji *fit_automl* mogłaby być kontynuowana w pozostałych funkcjach. Docstringi dodane 3 maja nie tłumaczą zbyt wiele.

Dla reprodukowalności wyników i możliwości skorzystania z całej zaimplementowanej funkcjonalności (również tej w notatniku) warto byłoby zamieścić informację jak otrzymać pliki .PKL, z których autorzy otrzymali wyniki w swoim notatniku.

2.7 Czy udało się odtworzyć zamieszone przykłady w kodzie?

Niestety nie, gdyż brakuje plików .PKL, które wykorzystywane są jako źródło danych. Nie oznacza to jednak tego, że nie udało się sprawdzić poprawności działania głównej części kodu. O tym w kolejnym podpunkcie.

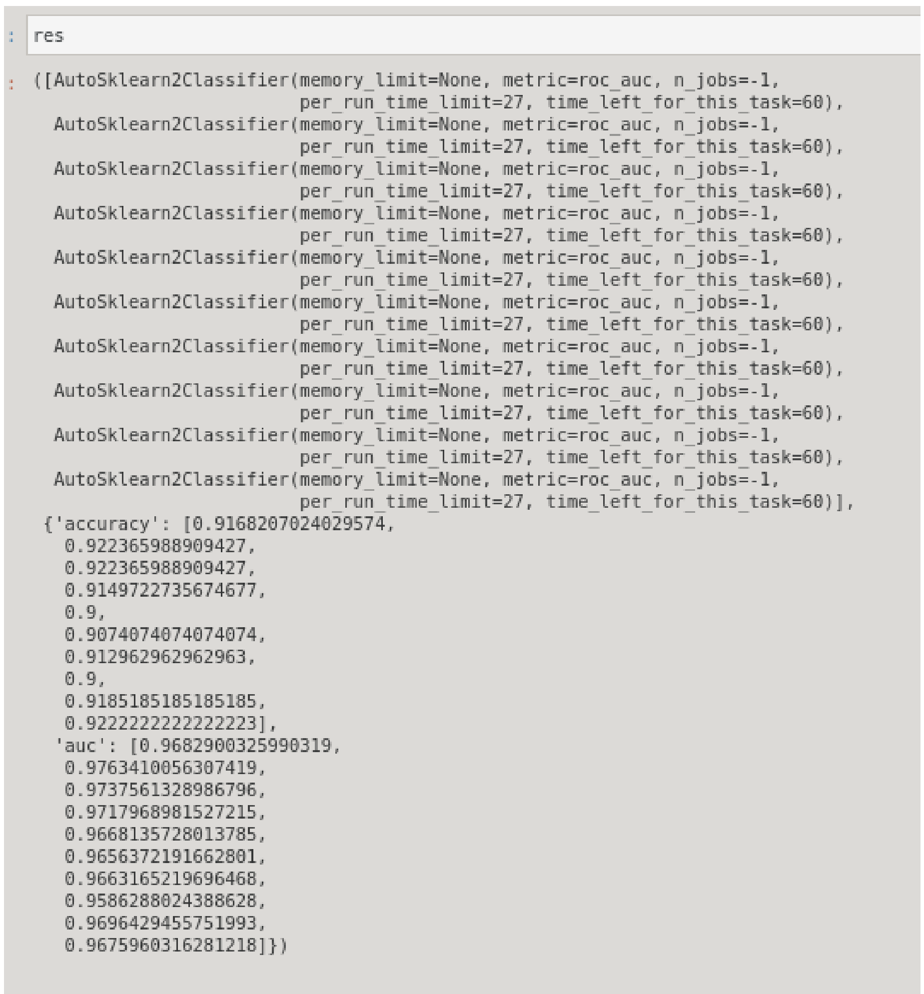
2.8 Czy udało się użyć przygotowanych kodów na nowych danych?

Tak (przynajmniej kodu zawartego w części zawartej w pliku .PY). Przedstawiają to Rysunki 1 oraz 2 dla wersji sprzed 3 maja oraz Rysunki 6 oraz 7 dla wersji po 3 maja.

Wykorzystano w tym celu zbiór danych *phoneme* https://www.openml.org/search?type=data&sort=runs&status=active&qualities.NumberOfClasses=%3D_2&id=1489

```
res = funkcja( X, pd.DataFrame(y3))
```

Rysunek 6: Wywołanie funkcji *funkcja* dostępnej w pliku .PY po 3 maja



```
res

([AutoSklearn2Classifier(memory_limit=None, metric=roc_auc, n_jobs=-1,
                         per_run_time_limit=27, time_left_for_this_task=60),
  AutoSklearn2Classifier(memory_limit=None, metric=roc_auc, n_jobs=-1,
                         per_run_time_limit=27, time_left_for_this_task=60),
  AutoSklearn2Classifier(memory_limit=None, metric=roc_auc, n_jobs=-1,
                         per_run_time_limit=27, time_left_for_this_task=60),
  AutoSklearn2Classifier(memory_limit=None, metric=roc_auc, n_jobs=-1,
                         per_run_time_limit=27, time_left_for_this_task=60),
  AutoSklearn2Classifier(memory_limit=None, metric=roc_auc, n_jobs=-1,
                         per_run_time_limit=27, time_left_for_this_task=60),
  AutoSklearn2Classifier(memory_limit=None, metric=roc_auc, n_jobs=-1,
                         per_run_time_limit=27, time_left_for_this_task=60),
  AutoSklearn2Classifier(memory_limit=None, metric=roc_auc, n_jobs=-1,
                         per_run_time_limit=27, time_left_for_this_task=60),
  AutoSklearn2Classifier(memory_limit=None, metric=roc_auc, n_jobs=-1,
                         per_run_time_limit=27, time_left_for_this_task=60),
  AutoSklearn2Classifier(memory_limit=None, metric=roc_auc, n_jobs=-1,
                         per_run_time_limit=27, time_left_for_this_task=60),
  AutoSklearn2Classifier(memory_limit=None, metric=roc_auc, n_jobs=-1,
                         per_run_time_limit=27, time_left_for_this_task=60)],
 {'accuracy': [0.9168207024029574,
   0.922365988909427,
   0.922365988909427,
   0.9149722735674677,
   0.9,
   0.9074074074074074,
   0.912962962962963,
   0.9,
   0.9185185185185185,
   0.9222222222222223],
  'auc': [0.9682900325990319,
   0.9763410056307419,
   0.9737561328986796,
   0.9717968981527215,
   0.9668135728013785,
   0.9656372191662801,
   0.9663165219696468,
   0.9586288024388628,
   0.9696429455751993,
   0.9675960316281218]})
```

Rysunek 7: Rezultat otrzymany z funkcji *funkcja*

3 Podsumowanie

- W przypadku błędu podczas załączania paczek warto upewnić się, czy ich wersje są zgodne z używaną wersją auto-sklearn. Na dzień 3 maja, auto-sklearn w najnowszej wersji działał dobrze z scipy 1.7.3 (była to jedyna paczka wymagająca ingerencji, gdyż 2 maja wyszła wersja 1.8.0, z którą kod nie działał poprawnie)
- Kod działa poprawnie, choć uzyskanie przykładu działającego rezultatu wymaga nieco dodatkowej pracy (po zmianie z dnia 3 maja, dodatkowa praca jest już znikoma)
- Nie ma żadnych łatwo widocznych błędów logicznych, ani prostych metod na przyspieszenie kodu
- Styl kodu jest w większości zgodny z PEP 8, choć występuje kilka odstępstw od tego
- Dokumentacja mogłaby być nieco bogatsza, w szczególności w postaci dodania docstringów lub poszerzenia informacji w nich zawartych (jeśli już są obecne)