Code review - autoPyTorch

Piotr Marciniak

Podstawowe informacje

Funkcja ta osiąga cel. Przyjmuje jako argumenty X, y i parametry dla frameworku oraz zwraca modele wraz z wynikami dla danych foldów.

Nie widzę w nim żadnych poważnych błędów logicznych. Problematyczny wydaje mi się brak mieszania danych przed podziałem ich na foldy. Może to sprawić, że pewne specyficzne przypadki trafią do tych samych foldów. Jednak funkcja może zakładać, że dane przed podziałem na foldy są mieszane. Oprócz tego zastanawia mnie fakt, że w kodzie nie ma żadnego preprocessingu danych oprócz zamiany typu kolumny object na category. W prezentacji nie było nic na temat preprocessingu dat. Na podstawie pierwszej prezentacji wydaję mi się, że w tym przypadku wszystkie daty byłyby zakodowane One-Hot-Enkoderem, co nie wydaje mi się dobrym pomysłem. Oczywiście ten preprocessing nie jest problemem z naszym benchmarkiem, ponieważ nie było w nim żadnych dat.

Wydaje mi się, że wszystkie wymagania są zaimplementowane. Jedyną modyfikacja danych wejściowych przed skorzystaniem z frameworka jest zamiana kolumn o typie object do typu category, ponieważ autoPyTorch radzi sobie z preprocessingiem danych kategorycznych i numerycznych. Jako, że framework nie wspierał żadnego przekazywania splitera, podział danych jest zaimplementowany na zewnątrz.

Uwagi stylistyczne

Białe linie

- Jak widzimy poniżej w listingu 1 jest tylko jedna linia biała po importach, a powinny być dwie. Jest to niezgodne z PEP 8: E302.

```
import os
import pandas as pd
from autoPyTorch.api.tabular_classification import TabularClassificationTask
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import StratifiedKFold
import tempfile as tmp
import warnings

def funkcja(X,y, fold_time = 300, time_per_model = 75, n_folds = 10):
```

Listing 1: PEP 8: E302

- Natomiast w listingu numer 2 widzimy dwie białe linie w środku funkcji. Według PEP 8 powinna być tam maksymalnie 1 linia stanowiąca logiczne rozdzielenie kodu.

```
    warnings.simplefilter(action='ignore', category=UserWarning)
    warnings.simplefilter(action='ignore', category=FutureWarning)



    for feature in X.columns:
        if X[feature].dtype == 'object':
            X[feature] = X[feature].astype('category')
```

```
    if y.dtype == 'object':
        y = y.astype('category')
```

Listing 2: PEP 8: E303

Generalnie dwie linie kodu dajemy pomiędzy znaczącymi wiele blokami kodu (importy, definicje funkcji).

Spacje

- Zgodnie z PEP 8: E231 podając argumenty funkcji powinna być spacja po przecinku.

```
    def funkcja(X,y, fold_time = 300, time_per_model = 75, n_folds = 10):
```

Listing 3: PEP 8: E231

- Zgodnie z PEP 8: E225 powinna być spacja wokół operatora

```
        X_train= X.iloc[train_idx]
        y_train= y.iloc[train_idx]
        X_test = X.iloc[test_idx]
        y_test = y.iloc[test_idx]
```

Listing 4: PEP 8: E225

- Zgodnie z PEP 8: E251 niepowinno być spacji, gdy przypisujemy wartości argumentom

```
api = TabularClassificationTask()
api.search(X_train= X_train, y_train= y_train,X_test = X_test,
            y_test = y_test, optimize_metric='roc_auc',total_walltime_limit=fold_time,
            func_eval_time_limit_secs=time_per_model,  memory_limit=None)
```

Listing 5: PEP 8: E251 (mamy połamane niektóre linie aby je zmieścić na stronie)

Co więcej PEP8 sugeruje aby długość linii nie przekraczała 79 znaków, co jest nie spełniony w powyższym kodzie.

Nazywanie zmiennych

Nazwy zmiennych w funkcji powinny być z małej litery zgodnie z PEP8.

```
SKF = StratifiedKFold(n_splits=n_folds)
```

Listing 6: Nazwy zmiennych

Rozwiązanie

Najszybszym rozwiązaniem, jakie przychodzi mi na myśl, jest skorzystanie z PyCharma. Jeśli chodzi o spację i puste linie możemy skorzystać z reformat code (skrót klawiszowy ctrl+alt+l). Natomiast jeśli chodzi o nazwy zmiennych można skorzystać ze skrótu klawiszowego shift+f6, który pozwala zamienić daną nazwę w ramach danego bloku kodu.

Sugerowane poprawki kodu

Zgodnie z PEP 8 porównaniu obiektów powinno zawsze używać **isinstance()** zamiast porównywania bezpośredniego.

```
# Correct:
if isinstance(obj, int):
# Wrong:
if type(obj) is type(1):
```

Listing 7: Przykład z PEP 8

Dlatego wydaję mi się, że lepszym porównaniem typów kolumn niż te w listingu 8 byłoby skorzystanie z funkcji pandasa.

```
    for feature in X.columns:
        if X[feature].dtype == 'object':
            X[feature] = X[feature].astype('category')
    if y.dtype == 'object':
        y = y.astype('category')
```

Listing 8: Orginalny kod

Chodzi mi konkretnie o skorzystanie z funkcji wbudowanych w pandasa takich jak moduł **pandas.api.types**, który zawiera takie funkcje jak **is_string_dtype**, **is_numeric_dtype**. Zastosowanie takich funkcji poprawiłoby także czytelność kodu. Inną opcją jest skorzystanie z metody **select_dtypes** dla DataFrame'u, która wybiera podzbiór kolumn z danym typem danych. Skorzystanie z **select_dtypes** dla DataFrame'u pozwoliło by skrócić kod ponieważ nie musielibyśmy pisać pętli. Chociaż nie jest to zbyt duże skrócenie długości kodu.

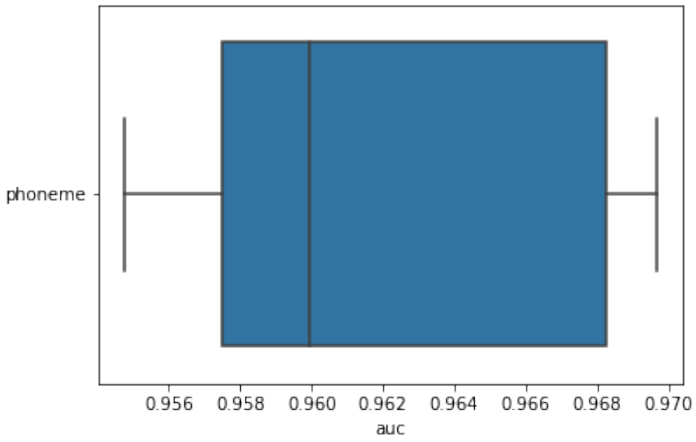
Dokumentacja i komentarze

Chociaż w kodzie nie ma żadnej dokumentacji i komentarzy nie było to dla mnie problematyczne. Nazwy zmiennych są w większości samowyjaśnialne oraz kod użyty w funkcji jest dosyć prosty. Z drugiej strony nie obraziłbym się, gdyby funkcja miała docstringa opisującego, jakiego X, y oczekuje i co zwraca.

Uruchomienie kodów

Przykłady w kodzie

Niestety nie da się odtworzyć przykładów w kodzie przy użyciu tej funkcji, ponieważ funkcja w środku tworzy sobie spliter. Z prezentacji i kodu wiem, że zespół korzystał z podanych indeksów przez OpenML'a oraz z trochę innej implementacji funkcji, a nie dokładnie tej funkcji. Przy zadanych takich samych warunkach startowych dla frameworku udało się osiągnąć poniższe wyniki dla zbioru **phoneme**.



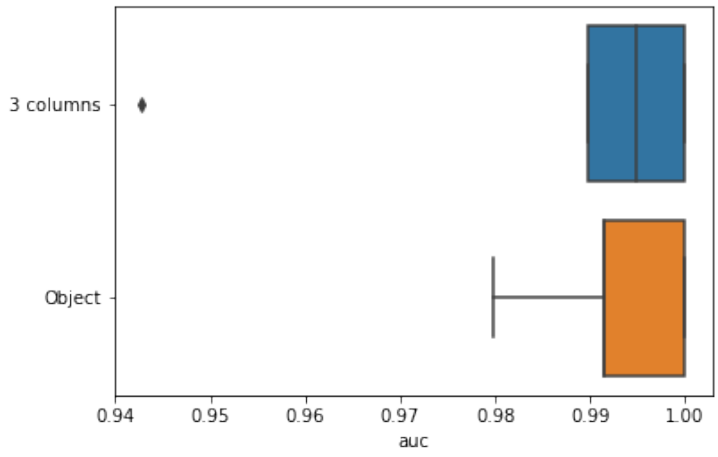
Rysunek 1: Wyniki dla zbioru phoneme

Z przedstawionych wykresów na prezentacji działania frameworku wydaje mi się, że wyniki są dosyć podobne.

Nowe zbiory

Algerian Forest Fires Dataset Data Set

Pierwszym zbiorem, na którym uruchomiłem framework to zbiór Algerian Forest Fires Dataset Data Set. Jest to dosyć mały zbiór składający się tylko z 244 obserwacji, zawierający kolumny liczbowe i datę, która jest już sformatowana (mamy 3 kolumnę oznaczający rok, miesiąc i dzień). Próbujemy przewidzieć, czy las danego dnia był w ogniu, czy nie. Jeśli podajemy kolumnę jako typ datetime, funkcja wyrzuca błąd nie uruchamia się. Poniżej przedstawiam dwa podejścia raz kolumna z datą została przekazana jako object, natomiast drugi raz została zmieniona do 3 kolumn, które mówiły jakiego dnia, miesiąca, roku dotyczy obserwacja.

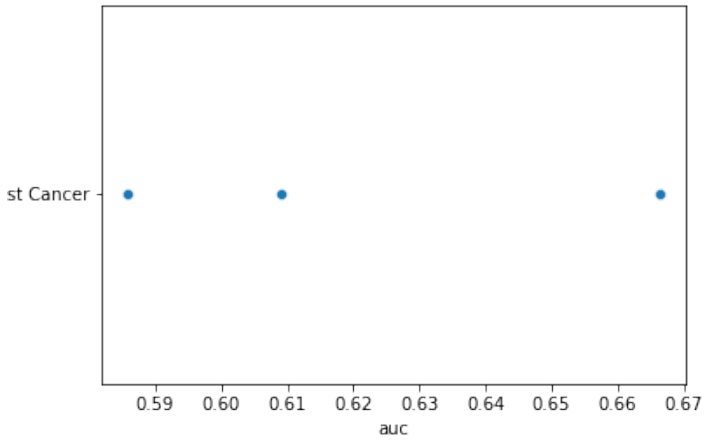


Rysunek 2: Wyniki dla zbioru Algerian Forest Fires

Breast Cancer Data Set

Drugim zbiorem, na którym sprawdzałem autoPyTorcha, był zbiór Breast Cancer Data Set. Jest to zbiór niezbilansowany zawierający 201 obserwacji jednej klasy oraz 85 obserwacji innej klasy. Klasy są opisane przez

zmienne kategoryczne oraz przez przedziały wartości (zbiór zawiera braki danych). Próbujemy przewidzieć, czy wydarzenie się powtarza, czy nie. Poniżej mamy wyniki dla tego zbioru, aby je osiągnąć musiałem zmniejszyć czas na folda i liczbę foldów, ponieważ w innym przypadku był wyrzucany błąd przedstawiony na obrazku 4.



Rysunek 3: Wyniki dla zbioru Breast Cancer

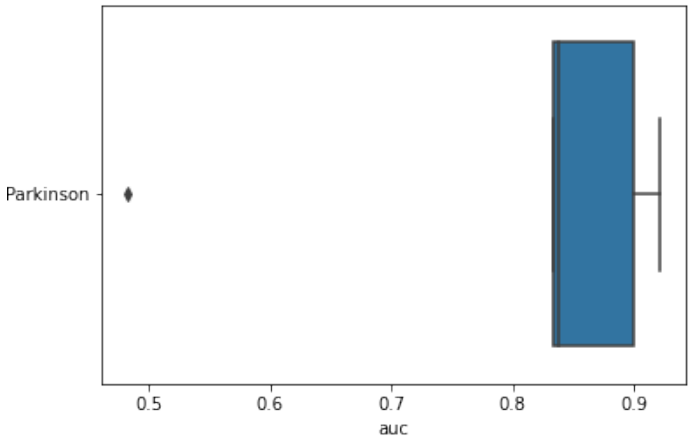
```
File /opt/conda/envs/autopytorch/lib/python3.8/multiprocessing/context.py:83, in BaseContext.Semaphore(self, value)
     81 '''Returns a semaphore object'''
     82 from .synchronize import Semaphore
---> 83 return Semaphore(value, ctx=self.get_context())

File /opt/conda/envs/autopytorch/lib/python3.8/multiprocessing/synchronize.py:126, in Semaphore.__init__(self, value, ctx)
    125 def __init__(self, value=1, *, ctx):
--> 126     SemLock.__init__(self, SEMAPHORE, value, SEM_VALUE_MAX, ctx=ctx)

File /opt/conda/envs/autopytorch/lib/python3.8/multiprocessing/synchronize.py:57, in SemLock.__init__(self, kind, value, maxvalue, ctx)
     55 for i in range(100):
     56     try:
---> 57         sl = self._semlock = _multiprocessing.SemLock(
     58             kind, value, maxvalue, self._make_name(),
     59             unlink_now)
     60     except FileExistsError:
     61         pass

OSError: [Errno 28] No space left on device
```

Rysunek 4: Pojawiający się błąd

Błąd ten może zależeć od tego, że kod był puszczany na dostępnej nam maszynie. Z tego co znalazłem w internecie nie koniecznie on oznacza małą liczbę miejsca na dysku. Widziałem, że może on oznaczać, że folder *tmp* może być wyczerpany. Nie wydaję mi się, że on zależy od implementacji funkcji.

Parkinson

W związku z problemami z drugim zbiorem postanowiłem sprawdzić wyniki na nowym zbiorze Parkinson, na którym pojawił się także ten sam błąd. Po zmianie liczby foldów i czasów błąd się nie pojawił. Wyniki poniżej



Rysunek 5: Wyniki dla zbioru Parkinson

Podsumowanie

Podsumowując

- kod osiąga postawiony cel,
- nie ma żadnych poważnych błędów logicznych (chociaż wydaję mi, że są pewne luki w rozumowaniu),
- kod nie jest zgodny z PEP 8,
- praktycznie nie ma obszarów, w których kod można poprawić (zasugerowałem zmiany bardziej poprawiające czytelność i niewiele skracające kod),
- dokumentacji i komentarzy nie ma, ale nie przeszkadza to bardzo w zrozumieniu kodu,
- nie udało się przy użyciu funkcji, ale po puszczeniu na jednym z zbiorów osiągnąłem podobne wyniki,
- udało się użyć przygotowanych kodów na 3 nowych zbiorach danych.