

# FLAML

Grupa KTR

WB-2022-AUTOML

07.04.2022

## Outline

- Wprowadzenie
- Czym jest FLAML?
- możliwości
- zawartość biblioteki
- braki
- dostępne modele
- integracje

AutoML

`_init_`

obsługa

`**settings`

Optymalizacja hiperparametrów

`default`

## Czym jest FLAML?



### A Fast Library for Automated Machine Learning & Tuning

Biblioteka do Pythona tworzona przez Microsoft, która automatycznie znajduje optymalne modele uczenia maszynowego w sposób efektywny i ekonomiczny. Pozwala użytkownikom pominąć etap wybierania odpowiednich modelów oraz dobierania dla każdego z nich hiperparametrów.

## Możliwości

- ▶ Dla często występujących problemów takich jak klasyfikacja czy regresja, szybko znajduje jakościowe modele przy niskim użyciu zasobów obliczeniowych.
- ▶ Wspiera zarówno klasyczne modele uczenia maszynowego jak i głębokie sieci neuronowe.
- ▶ Łatwy do dostosowania i rozszerzenia.
- ▶ Wspiera szybką i wydajną automatyczną optymalizację zdolną do działania na dużej przestrzeni poszukiwań.
- ▶ Istnieje jej implementacja w .NET.

## Zawartość biblioteki

- ▶ default
- ▶ nlp
- ▶ onlineml
- ▶ scheduler
- ▶ tune
- ▶ automl
- ▶ data
- ▶ ml
- ▶ model

## Braki

- biblioteka nie oferuje preprocessingu danych

## Dostępne modele

- ▶ LGBM
- ▶ XGBoost
- ▶ ExtraTrees
- ▶ randomforest
- ▶ catboost
- ▶ KNeighbors
- ▶ i inne - większość z nich zarówno do klasyfikacji jak i regresji
- ▶ także możliwość dodania swojego modelu

## integracje

flaml pozwala na użycie dowolnego modelu, ze szczególną łatwością użycia estymatorów sklearnowych, czyli z metodami `.fit()` i `.predict()`

## AutoML

### AutoML

Główna klasą biblioteki jest `flaml.AutoML`.  
jej konstruktor przyjmuje kwarg nazwany 'settings' argumenty  
podane w konstruktorze mogą być nadpisane przy wywołaniu  
metody `AutoML.fit()`

- ▶ trenuje modele
- ▶ optymalizuje hiperparametry

## obsługa

### kompatybilność z sklearnem

tak jak w sklearnie fituje się dane metodą `.fit()` oraz klasa AutoML udostępnia metody `.predict_proba()` oraz `.predict()`

## obsługa

### `fit()`

Wykonując metodę `fit()` jednocześnie trenujemy dane na kilku modelach stosowanych do naszego typu problemu oraz optymalizujemy dla nich hiperparametry. Możemy uzyskać najlepszy model oraz konfigurację hiperparametrów dzięki metodom `best_estimator` oraz `best_config`.

W `fit()` podajemy X oraz y, gdyż mamy do czynienia z uczeniem nadzorowanym.

## \*\*settings

wybrane najważniejsze ustawienia:

- ▶ metric - 'roc\_auc', 'accuracy' itd.
- ▶ task - 'classification', 'regression', 'ts\_forecast', 'rank',  
'seq-classification', 'seq-regression', 'summarization'
- ▶ estimator\_list - 'auto', 'lgbm', 'xgboost', 'catboost', 'rf'
- ▶ time\_budget - czas w sekundach

## przykłady tasków

```
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
x,y = make_classification()
x_train,x_test,y_train,y_test = train_test_split(
    x,y,stratify=y,
    test_size=.3)

ml = AutoML()
ml.fit(x_train,y_train)
accuracy_score(y_test,ml.predict(x_test))
```

## przykłady tasków II

ważne elementy outputu:

```
INFO - task = classification
INFO - Data split method: stratified
INFO - Evaluation method: cv
INFO - Minimizing error metric: 1-roc_auc
INFO - List of ML learners in AutoML Run: ['lgbm', 'rf',
'xgboost', 'extra_tree', 'xgb_limitdepth', 'lrl1']
```

0.8666666666666667

model spoza flam

```
from flaml import AutoML
from sklearn.svm import SVR
from flaml.tune import uniform
from flaml.model import SKLearnEstimator
from sklearn.datasets import make_regression
x,y = make_regression()
```

model spoza flaml

```
class flamlsvr(SKLearnEstimator):
    def __init__(self, task = "regression", **config):
        config.pop('n_jobs')
        super().__init__(task, **config)
        self.estimator_class = SVR

    @classmethod
    def search_space(cls, data_size, task):
        space = {
            'C':{
                'domain':uniform(1e-10,1),
                'low_cost_init_value':.2
            }
        }
        return space
```

model spoza flaml

```
ml = AutoML()
ml.add_learner("svr",famlsvr)
ml.fit(x,y,
**{'task': 'regression', 'estimator_list':['svr']},
time_budget=15)
```

## tune

`f1aml.tune` jest modelem pakietu `f1aml`. Jest on nastawiony na optymalizację hiperparametrów w ekonomiczny sposób. Może być używany zarówno wewnątrz `f1aml`, `AutoML`, jak i osobno.

## Procedura tuningu

- ▶ Sprecyzowanie jaki jest cel tuningu
- ▶ Podanie przestrzeni hiperparametrów, w których mamy znaleźć optymalne
- ▶ Podanie ograniczeń tuningu

## tune.run()

Po wykonaniu czynności z poprzedniego slajdu wykonujemy metodę `f1aml.tune.run()` i rozpoczętym proces optymalizacji. Zwracany zostaje obiekt klasy `ExperimentalAnalysis`, która może nam zwrócić najlepszą konfigurację parametrów.

## Działanie

Do optymalizacji hiperparametrów FLAML wykorzystuje dwie techniki:

- ▶ BlendSearch (metoda domyślna)
- ▶ CFO

## **f1aml.default**

f1aml.default pozwala na uzyskanie dobrej konfiguracji hiperparametrów bez wykonywania czasochłonnej optymalizacji.  
Działanie jest bardzo proste, wystarczy zamienić  
import <nazwa modelu> from <nazwa pakietu>  
na from f1aml.default import <nazwa pakietu>