AutoML Model Creation(select the best model)

October 3, 2021

# AutoML Model Creation(select the best model)

1. **Tree-based Pipeline Optimization Tool (TPOT)**

TPOT is built on top of scikit-learn. TPOT uses a genetic algorithm to search for the best model according to the “generations” and “population size”. The higher the two parameters are set, the longer will it take time. Unlike AutoSklearn, we do not set the specific running time for TPOT. As its name suggests, after the TPOT is run, it exports lines of code containing a pipeline from importing packages, splitting the dataset, creating the tuned model, fitting the model, and finally predicting the validation dataset. The pipeline is exported in .py format.

In the code below, I set the generation and population\_size to be 5. The output gives 5 genera- tions with increasing “scoring”. I set the scoring to be “neg\_mean\_absolute\_error” and” accuracy” for regression and classification tasks respectively. Neg\_mean\_absolute\_error means Mean Abso- lute Error (MAE) in negative form. The algorithm chooses the highest scoring value. Making the MAE negative will make the algorithm selecting the MAE closest to zero.

Regression

from tpot import TPOTRegressor # Create model cv = RepeatedStratifiedKFold(n\_splits=3, n\_repeats=3, random\_state=123) tpot = TPOTRegressor(generations=5, population\_size=5, cv=cv, scoring=‘neg\_mean\_absolute\_error’, verbosity=2, random\_state=123, n\_jobs=-1) # Fit the training data tpot.fit(X\_train, y\_train) # Export the result tpot.export(‘tpot\_model.py’) Output:

Generation 1 - Current best internal CV score: -20390.588131563232 Generation 2 - Cur- rent best internal CV score: -19654.82630417806 Generation 3 - Current best internal CV score: -19312.09139004322 Generation 4 - Current best internal CV score: -19312.09139004322 Generation 5 - Current best internal CV score: -18752.921100941825 Best pipeline: Ran- domForestRegressor(input\_matrix, bootstrap=True, max\_features=0.25, min\_samples\_leaf=3, min\_samples\_split=2, n\_estimators=100) Classification

from tpot import TPOTClassifier # TPOT that are stopped earlier. It still gives tempo- rary best pipeline. # Create the model cv = RepeatedStratifiedKFold(n\_splits=3, n\_repeats=3, random\_state=123) tpot = TPOTClassifier(generations=5, population\_size=5, cv=cv, scor- ing=‘accuracy’, verbosity=2, random\_state=123, n\_jobs=-1) # Fit the training data tpot.fit(X\_train, y\_train) # Export the result tpot.export(‘tpot\_model.py’) Output:

Generation 1 - Current best internal CV score: 0.7432273262661955 Generation 2

- Current best internal CV score: 0.843824979278454 Generation 3 - Current best in- ternal CV score: 0.8545565589146273 Generation 4 - Current best internal CV score: 0.8545565589146273 Generation 5 - Current best internal CV score: 0.859616978580465 Best pipeline: RandomForestClassifier(GradientBoostingClassifier(input\_matrix, learn- ing\_rate=0.001, max\_depth=2, max\_features=0.7000000000000001, min\_samples\_leaf=1,

min\_samples\_split=19, n\_estimators=100, subsample=0.15000000000000002), bootstrap=True, criterion=gini, max\_features=0.8500000000000001, min\_samples\_leaf=4, min\_samples\_split=12, n\_estimators=100) AutoSklearn gives the result of RandomForestRegressor for the regres- sion task. As for the classification, it gives the stacking of GradientBoostingClassifier and RandomForestClassifier. All algorithms already have their hyperparameters tuned.

Here is to see the validation data scoring metrics.

Regression

pred\_tpot = results # Scatter plot true and predicted values plt.scatter(pred\_tpot, y\_val, alpha=0.2) plt.xlabel(‘predicted’) plt.ylabel(‘true value’) plt.text(100000, 400000, ‘RMSE:’ + str(round(MSE(y\_val, pred\_tpot)\*\*0.5))) plt.text(100000, 350000, ‘MAE:’

+ str(round(mean\_absolute\_error(y\_val, pred\_tpot)))) plt.text(100000, 300000, ‘R:’ + str(round(np.corrcoef(y\_val, pred\_tpot)[0,1],4))) plt.show() Output:

TPOT Fig. 2 TPOT regression result.

Classification

pred\_tpot = results # Compute the accuracy print(‘Accuracy:’ + str(accuracy\_score(y\_val, pred\_tpot))) print(’‘) # Prediction results print(’Confusion Matrix’) print(pd.DataFrame(confusion\_matrix(y\_val, pred\_tpot), index=[1,2,3,4], columns=[1,2,3,4])) print(’‘) print(’Classification Report’) print(classification\_report(y\_val, pred\_tpot)) Output:

Accuracy: 0.9246467817896389

Confusion Matrix 1 2 3 4 1 117 11 7 16 2 6 288 10 15 3 2 18 186 36 4 5 12 6 1176

Classification Report precision recall f1-score support

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 1 | 0.90 | 0.77 | 0.83 | 151 |
| 2 | 0.88 | 0.90 | 0.89 | 319 |
| 3 | 0.89 | 0.77 | 0.82 | 242 |
| 4 | 0.95 | 0.98 | 0.96 | 1199 |
| accuracy |  |  | 0.92 | 1911 |

macro avg 0.90 0.86 0.88 1911 weighted avg 0.92 0.92 0.92 1911

# AutoML with FLAML Library

FLAML is a Python library (https://github.com/microsoft/FLAML) designed to automatically produce accurate machine learning models with low computational cost. It is fast and cheap. The simple and lightweight design makes it easy to use and extend, such as adding new learners.

FLAML can

serve as an economical AutoML engine, be used as a fast hyperparameter tuning tool

-> pip install flaml[notebook] #Classification Example

from flaml.data import load\_openml\_dataset X\_train, X\_test, y\_train, y\_test = load\_openml\_dataset(dataset\_id=1169, data\_dir=‘./’) load dataset from ./openml\_ds1169.pkl Dataset name: airlines X\_train.shape: (404537, 7), y\_train.shape: (404537,); X\_test.shape: (134846, 7), y\_test.shape: (134846,) Run FLAML In the FLAML automl run configuration, users can specify the task type, time budget, error metric, learner list, whether to subsample, resampling strategy type, and so on. All these arguments have default values which will be used if users do not provide them. For example, the default ML learners of FLAML are [‘lgbm’, ‘xgboost’, ‘catboost’, ‘rf’, ‘extra\_tree’, ‘lrl1’].

[ ]:

In [2]: ”’ import AutoML class from flaml package ”’ from flaml import AutoML automl = AutoML() In [3]: settings = { “time\_budget”: 240, # total running time in seconds “metric”: ‘ac- curacy’, # can be: ‘r2’, ‘rmse’, ‘mae’, ‘mse’, ‘accuracy’, ‘roc\_auc’, ‘roc\_auc\_ovr’, # ‘roc\_auc\_ovo’, ‘log\_loss’, ‘mape’, ‘f1’, ‘ap’, ‘ndcg’, ‘micro\_f1’, ‘macro\_f1’ “task”: ‘classification’, # task type “log\_file\_name”: ‘airlines\_experiment.log’, # flaml log file “seed”: 7654321, # random seed } In [4]: ’‘’The main flaml automl API”’ automl.fit(X\_train=X\_train, y\_train=y\_train, \*\*settings) [flaml.automl: 08-31 00:53:33] {1279} INFO - Evaluation method: holdout [flaml.automl: 08-31

00:53:34] {1312} INFO - Minimizing error metric: 1-accuracy [flaml.automl: 08-31 00:53:34] {1338} INFO - List of ML learners in AutoML Run: [‘lgbm’, ‘rf’, ‘catboost’, ‘xgboost’, ‘extra\_tree’, ‘lrl1’]

Best model and metric In [5]: ”’ retrieve best config and best learner”’ print(‘Best ML leaner:’, automl.best\_estimator) print(‘Best hyperparmeter config:’, automl.best\_config) print(‘Best accu- racy on validation data: {0:.4g}’.format(1-automl.best\_loss)) print(‘Training duration of best run:

{0:.4g} s’.format(automl.best\_config\_train\_time)) Best ML leaner: lgbm

Best hyperparmeter config: {‘n\_estimators’: 220, ‘num\_leaves’: 270, ‘min\_child\_samples’: 4,

‘learning\_rate’: 0.04886499949999022, ‘log\_max\_bin’: 10, ‘colsample\_bytree’: 0.3841266992710469,

‘reg\_alpha’: 0.0009765625, ‘reg\_lambda’: 0.07539015928723636, ‘FLAML\_sample\_size’: 364083}

Best accuracy on validation data: 0.67 Training duration of best run: 9.323 automl.model.estimator Out[6]: LGBMClassifier(colsample\_bytree=0.3841266992710469,

learning\_rate=0.04886499949999022, max\_bin=512, min\_child\_samples=4, n\_estimators=220, num\_leaves=270, objective=‘binary’, reg\_alpha=0.0009765625, reg\_lambda=0.07539015928723636, verbose=-1) In [7]: ”’ pickle and save the automl object ”’ import pickle with open(‘automl.pkl’, ‘wb’) as f: pickle.dump(automl, f, pickle.HIGHEST\_PROTOCOL) In [8]: ”’ compute predictions of testing dataset ”’ y\_pred = automl.predict(X\_test) print(‘Predicted labels’, y\_pred) print(‘True labels’, y\_test) y\_pred\_proba = automl.predict\_proba(X\_test)[:,1] Predicted labels [‘1’ ‘0’ ‘1’ . . . ‘1’ ‘0’ ‘0’] True labels 118331 0 328182 0

compute different metric values on testing dataset

from flaml.ml import sklearn\_metric\_loss\_score print(‘accuracy’, ‘=’, 1 - sklearn\_metric\_loss\_score(‘accuracy’, y\_pred, y\_test)) print(‘roc\_auc’, ‘=’, 1 - sklearn\_metric\_loss\_score(‘roc\_auc’, y\_pred\_proba, y\_test)) print(‘log\_loss’, ‘=’, sklearn\_metric\_loss\_score(‘log\_loss’, y\_pred\_proba, y\_test)) accuracy = 0.6729231864497278 roc\_auc = 0.7261961112785199 log\_loss = 0.6033707263741326

See Section 4 for an accuracy comparison with default LightGBM and XGBoost.

Log history

from flaml.data import get\_output\_from\_log time\_history, best\_valid\_loss\_history, valid\_loss\_history, config\_history, metric\_history = get\_output\_from\_log(filename=settings[‘log\_file\_name’], time\_budget=240) for config in con- fig\_history: print(config) {‘Current Learner’: ‘lgbm’, ‘Current Sample’: 10000, ‘Current Hyper- parameters’: {‘n\_estimators’: 4, ‘num\_leaves’: 4, ‘min\_child\_samples’: 20, ‘learning\_rate’:

0.09999999999999995, ‘log\_max\_bin’: 8, ‘colsample\_bytree’: 1.0, ‘reg\_alpha’: 0.0009765625, ‘reg\_lambda’: 1.0, ‘FLAML\_sample\_size’: 10000}, ‘Best Learner’: ‘lgbm’, ‘Best Hyper- parameters’: {‘n\_estimators’: 4, ‘num\_leaves’: 4, ‘min\_child\_samples’: 20, ‘learning\_rate’:

0.09999999999999995, ‘log\_max\_bin’: 8, ‘colsample\_bytree’: 1.0, ‘reg\_alpha’: 0.0009765625,

‘reg\_lambda’: 1.0, ‘FLAML\_sample\_size’: 10000}}

**import matplotlib.pyplot as plt import numpy as np**

[ ]:

plt.title('Learning Curve') plt.xlabel('Wall Clock Time (s)') plt.ylabel('Validation Accuracy')

plt.scatter(time\_history, 1 - np.array(valid\_loss\_history)) plt.step(time\_history, 1 - np.array(best\_valid\_loss\_history), where='post') plt.show()

*#Default LightGBM*

**from lightgbm import** LGBMClassifier lgbm = LGBMClassifier() lgbm.fit(X\_train, y\_train) LGBMClassifier()

y\_pred\_lgbm = lgbm.predict(X\_test)

*#Default XGBoost*

**from xgboost import** XGBClassifier xgb = XGBClassifier()

cat\_columns = X\_train.select\_dtypes(include=['category']).columns X = X\_train.copy()

X[cat\_columns] = X[cat\_columns].apply(**lambda** x: x.cat.codes) xgb.fit(X, y\_train)

XGBClassifier(base\_score=0.5, booster='gbtree', colsample\_bylevel=1, colsample\_bynode=1, colsample\_bytree=1, gamma=0, gpu\_id=-1, importance\_type='gain', interaction\_constraints='', learning\_rate=0.300000012, max\_delta\_step=0, max\_depth=6, min\_child\_weight=1, missing=nan, monotone\_constraints='()', n\_estimators=100, n\_jobs=0, num\_parallel\_tree=1, random\_state=0, reg\_alpha=0, reg\_lambda=1, scale\_pos\_weight=1, subsample=1, tree\_method='exact', validate\_parameters=1, verbosity=**None**)

X = X\_test.copy()

X[cat\_columns] = X[cat\_columns].apply(**lambda** x: x.cat.codes) y\_pred\_xgb = xgb.predict(X)

In [18]:

print('default xgboost accuracy', '=', 1 -␣

*‹→*sklearn\_metric\_loss\_score('accuracy', y\_pred\_xgb, y\_test))

print('default lgbm accuracy', '=', 1 - sklearn\_metric\_loss\_score('accuracy',␣

*‹→*y\_pred\_lgbm, y\_test))

print('flaml accuracy', '=', 1 - sklearn\_metric\_loss\_score('accuracy', y\_pred,␣

*‹→*y\_test))