AutoML Model Selection(best model select automatically)

October 3, 2021

1 AutoML Model Creation(select the best model)

2 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 generations 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 Absolute 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)
```

[ ]:
```
# 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 - Current 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: RandomForestRegressor(input_matrix, bootstrap=True, max_features=0.25, min_samples_leaf=3, min_samples_split=2, n_estimators=100)

```
[ ]: #classification
```

```
[ ]: # Compute the accuracy
print('Accuracy: ' + str(accuracy_score(y_val, pred_tpot)))
```

```
[ ]: # 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

3 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
```

```
[ ]:
```

```
[ ]: import AutoML class from flaml package
```

```
[ ]: #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'].
```

[ ]:
```
from flaml import AutoML automl = AutoML()
```

[ ]:
```
settings = { "time_budget": 240,
             total running time in seconds "metric": 'accuracy',
             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
```

[ ]:
```
#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))
```

[ ]: