# -\*-coding:utf-8 -\*-

import os

import json

import pickle

import numpy as np

import xgboost as xgb

import lightgbm as lgb

import matplotlib.pyplot as plt

from bayes\_opt import BayesianOptimization

from sklearn.metrics import mean\_absolute\_error, roc\_auc\_score, accuracy\_score, confusion\_matrix

class GBDTModel(object):

def \_\_init\_\_(self, output\_dir, n\_thread=10, n\_jobs=10, random\_state=7):

self.best\_model = None

self.best\_param = None

self.output\_dir = output\_dir

os.makedirs(output\_dir, exist\_ok=True)

self.n\_thread = n\_thread

self.n\_jobs = n\_jobs

self.random\_state = random\_state

self.clf\_dict = {

'lgb': lgb.LGBMClassifier,

'xgb': xgb.XGBClassifier

}

self.params\_dict = {

'lgb': {

'learning\_rate': (0.03, 0.2),

'n\_estimators': (100, 1000),

'num\_leaves': (20, 100),

'max\_depth': (2, 5),

'max\_bin': (30, 255),

'min\_data\_in\_bin': (30, 256),

'min\_data\_in\_leaf': (30, 500),

'min\_split\_gain': (0.1, 10),

'min\_child\_weight': (0.1, 100),

'min\_child\_samples': (30, 256),

'colsample\_bytree': (0.6, 1.0),

'reg\_alpha': (0.3, 10),

'reg\_lambda': (0.3, 10),

'subsample': (0.6, 1.0),

'subsample\_for\_bin': (200000, 400000),

'scale\_pos\_weight': (1.0, 1.5),

},

'xgb': {

'learning\_rate': (0.01, 0.3),

'n\_estimators': (100, 800),

'max\_depth': (2, 5),

'reg\_alpha': (0.00001, 5),

'reg\_lambda': (0.00001, 5),

'min\_child\_weight': (1, 100),

'subsample': (0.6, 1.0),

'colsample\_bytree': (0.6, 1.0),

'colsample\_bynode': (0.6, 1.0),

'colsample\_bylevel': (0.6, 1.0),

'gamma': (0.0, 0.3),

'scale\_pos\_weight': (1.0, 1.5)

}

}

self.int\_params = [

'max\_depth', 'n\_estimators', 'num\_leaves', 'max\_bin', 'scale\_pos\_weight',

'min\_child\_samples', 'min\_data\_in\_bin', 'min\_data\_in\_leaf', 'subsample\_for\_bin']

def save\_model(self, model\_name="model.pkl"):

file = os.path.join(self.output\_dir, model\_name)

with open(file, 'wb') as f:

pickle.dump(self.best\_model, f)

def read\_model(self, model\_name="model.pkl"):

file = os.path.join(self.output\_dir, model\_name)

with open(file, 'rb') as f:

self.best\_model = pickle.load(f)

def save\_best\_param(self, file\_name="params.json"):

file = os.path.join(self.output\_dir, file\_name)

with open(file, mode="w", encoding="utf-8") as fw:

json.dump(self.best\_param, fw, indent=4, ensure\_ascii=False)

def read\_best\_param(self, file\_name="params.json"):

file = os.path.join(self.output\_dir, file\_name)

with open(file=file, mode="r", encoding="utf-8") as fr:

self.best\_param = json.load(fr)

def bayes\_optimization(self, train\_x, train\_y, test\_x, test\_y, oot\_x=[], oot\_y=[],

clf\_name='lgb', target='acc', scale\_pos\_use=True, \*\*bayes\_args):

res\_list = []

n\_thread = self.n\_thread

n\_jobs = self.n\_jobs

random\_state = self.random\_state

int\_params = self.int\_params

xgb\_train = xgb.DMatrix(X\_train, y\_train)

xgb\_test = xgb.DMatrix(X\_test, y\_test)

if scale\_pos\_use is True:

scale\_pos\_weight = round(np.sum(train\_y == 0) / np.sum(train\_y == 1), 1)

if 'scale\_pos\_weight' in self.params\_dict[clf\_name].keys():

self.params\_dict[clf\_name]['scale\_pos\_weight'] = \

tuple([x \* scale\_pos\_weight for x in list(self.params\_dict[clf\_name]['scale\_pos\_weight'])])

else:

self.params\_dict[clf\_name]['scale\_pos\_weight'] = scale\_pos\_weight

else:

self.params\_dict[clf\_name]['scale\_pos\_weight'] = 1

print("[bayes\_optimization] scale\_pos\_weight: {}".format(

self.params\_dict[clf\_name]['scale\_pos\_weight']), flush=True)

def \_eval(\*\*clf\_args):

for param in int\_params:

if clf\_args.get(param):

clf\_args[param] = int(round(clf\_args[param]))

print("[eval] args: {}".format(clf\_args))

if clf\_name == 'lgb':

model = lgb.LGBMClassifier(\*\*clf\_args,

n\_jobs=n\_jobs,

random\_state=random\_state)

else:

model = xgb.XGBClassifier(\*\*clf\_args,

nthread=n\_thread,

n\_jobs=n\_jobs,

random\_state=random\_state)

model.fit(train\_x, train\_y)

# 绘制变量重要性

xgb.plot\_importance(model, height=0.5, importance\_type='gain', max\_num\_features=10)

plt.show()

# 交叉验证

result = xgb.cv(params=clf\_args, dtrain=xgb\_train, nfold=10, metrics='rmse', # 'auc'

num\_boost\_round=300, as\_pandas=True, seed=123)

print(result.head())

# Plot CV Errors

plt.plot(range(1, 301), result['train-rmse-mean'], 'k', label='Training Error')

plt.plot(range(1, 301), result['test-rmse-mean'], 'b', label='Test Error')

plt.xlabel('Number of Trees')

plt.ylabel('RMSE')

plt.axhline(0, linestyle='--', color='k', linewidth=1)

plt.legend()

plt.title('CV Errors for XGBoost')

plt.show()

train\_prob = model.predict\_proba(train\_x)[:, 1]

test\_prob = model.predict\_proba(test\_x)[:, 1]

#oot\_prob = model.predict\_proba(oot\_x)[:, 1]

train\_acc = round(accuracy\_score(train\_y, train\_prob > 0.5), 5)

train\_auc = round(roc\_auc\_score(train\_y, train\_prob), 5)

test\_acc = round(accuracy\_score(test\_y, test\_prob > 0.5), 5)

test\_auc = round(roc\_auc\_score(test\_y, test\_prob), 5)

#oot\_acc = round(accuracy\_score(oot\_y, oot\_prob > 0.5), 5)

#oot\_auc = round(roc\_auc\_score(oot\_y, oot\_prob), 5)

res\_list.append({

'train\_acc': train\_acc, 'train\_auc': train\_auc,

'test\_acc': test\_acc, 'test\_auc': test\_auc,

#'oot\_acc': oot\_acc, 'oot\_auc': oot\_auc

})

if target == 'auc':

target\_value = test\_auc

else:

target\_value = test\_acc

print('[train\_acc] {}, [train\_auc] {}, [test\_acc] {}, [test\_auc] {}'

#',[oot\_acc] {}, [oot\_auc] {}'

.format(

train\_acc, train\_auc, test\_acc, test\_auc

#,oot\_acc, oot\_auc

), flush=True)

return target\_value

print("[bayes\_optimization] {}".format(self.params\_dict[clf\_name]), flush=True)

clf\_bo = BayesianOptimization(f=\_eval, pbounds=self.params\_dict[clf\_name])

clf\_bo.maximize(\*\*bayes\_args)

self.best\_param = clf\_bo.max['params']

for param in int\_params:

if self.params\_dict[clf\_name].get(param):

self.best\_param[param] = int(round(self.best\_param[param]))

self.best\_param['nthread'] = n\_thread

self.best\_param['n\_jobs'] = n\_jobs

self.best\_param['random\_state'] = random\_state

res\_bo = []

for id, bo in enumerate(clf\_bo.res):

for param in int\_params:

if bo['params'].get(param):

bo['params'][param] = int(round(bo['params'][param]))

res\_bo.append([bo.update(res\_list[id])])

return clf\_bo, self.best\_param, res\_bo

def eval(self, x, y):

# 使用模型对测试集数据进行预测

predictions = self.best\_model.predict\_proba(x)[:, 1]

print("[MAE] {}".format(mean\_absolute\_error(y\_true=y, y\_pred=predictions)), flush=True)

print("[ACC] {}".format(accuracy\_score(y\_true=y, y\_pred=predictions > 0.5), flush=True))

print("[AUC] {}".format(roc\_auc\_score(y\_true=y, y\_score=predictions)), flush=True)

print("[confusion\_matrix] \n{} ".format(confusion\_matrix(y\_true=y, y\_pred=predictions > 0.5)), flush=True)

return predictions

def train(self, clf\_name, x, y):

print("[train] best\_param: {}".format(self.best\_param), flush=True)

if clf\_name == 'lgb':

print("[train] use model: LGBMClassifier", flush=True)

self.best\_model = lgb.LGBMClassifier(\*\*self.best\_param)

else:

print("[train] use model: XGBClassifier", flush=True)

self.best\_model = xgb.XGBClassifier(\*\*self.best\_param)

self.best\_model.fit(x, y)

print("[train] best\_model: {}".format(self.best\_model), flush=True)

self.save\_model()

import pandas as pd

from sklearn.preprocessing import LabelEncoder

data = pd.read\_csv(r"C:\Users\阿韩想养二哈\Desktop\数模经典数据集\PimaIndiansdiabetes.csv") # 输入特征

X = ['A','B','C','D','E','F','G','H']

#print(X)

y = ['I']

#print(y)

#此处为了后续输出混淆矩阵时，用原始数据输出

from sklearn.model\_selection import train\_test\_split

# 将数据分为训练数据和测试数据

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data[X], data[y], test\_size=0.3, train\_size=0.7, random\_state=42)

print(X\_train, X\_test, y\_train, y\_test)

use\_columns = pd.DataFrame(X)

label\_columns = pd.DataFrame(y)

train\_manager = GBDTModel(output\_dir="./")

#进行贝叶斯自动调参，并保存最优的参数。(数据均用pandas的DataFrame保存)

train\_manager.bayes\_optimization(

train\_x=X\_train, train\_y=np.array(y\_train),

test\_x=X\_test, test\_y=y\_test,

clf\_name="xgb"

)

#读取最优的参数，训练模型与评估

print('result:')

train\_manager.save\_best\_param(file\_name="params.json")

train\_manager.read\_best\_param(file\_name="params.json")

train\_manager.train(clf\_name="xgb",

x=X\_train, y=y\_train)

train\_manager.eval(x=X\_test, y=y\_test)

#train\_manager.eval(x=val\_data[use\_columns], y=val\_data[label\_columns])