#!/usr/bin/python3

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

"""

2020/11/19 by Owen Yang

##hz 修改机器学习验证集和测试集结果一致问题 ML\_Classfication

外部可引用函数:

特征分析选择:

func::principle\_component\_analysis:

func::feature importance:

regression based: Lasso, Ridge

classification based: xgboost(classifier)

func::model\_score\_with\_features\_add:

classification based: xgboost classifier, Logistic regression, SVC,

数据建模:

func::ML\_Classification:

'LogisticRegression',

'XGBClassifier',

'RandomForestClassifier',

'SVC',

'KNeighborsClassifier',

func::ML\_Regression:

'LinearRegression',

'XGBRegressor',

'RandomForestRegressor',

'LinearSVR',

'KNeighborsRegressor',

func::ML\_Clustering:

'KMeans',

'Birch',

'SpectralClustering',

'AgglomerativeClustering',

'GaussianMixture',

模型对比:

funct::two\_groups\_classfication\_multimodels:

'XGBClassifier'

'LogisticRegression',

'SVC',

'MLPClassifier',

'RandomForestClassifier',

'AdaBoostClassifier',

# 'KNeighborsClassifier',

# 'DecisionTreeClassifier',

# 'GradientBoostingClassifier',

# 'BaggingClassifier',

# 'ExtraTreesClassifier',

内部函数:

供 func::feature importance 使用:

func::\_lasso\_features\_importance

func::\_ridge\_features\_importance

func::\_xgboost\_features\_importance

自动寻参:

对于监督式学习(分类和回归)：

GridSearcherCV: 利用交叉验证为选择标准，对所有可能参数构型进行遍历搜索

RandSearcherCV: 利用交叉验证为选择标准，对所有可能参数构型进行随机搜索

对于非监督式学习(聚类)：

GridSearcherSelf: 利用给定模型选择标准(11/19：轮廓系数)，对所有可能参数构型进行随机搜索

RandSearcherSelf: 利用给定模型选择标准(11/19：轮廓系数)，对所有可能参数构型进行随机搜索

是否自动寻参作为每个函数的传递参量，默认为否。

目前支持自动寻参的函数有：

0. features\_importance (11/19)

1. ML\_Classification (11/19)

2. ML\_Regression (11/19)

3. ML\_Clustering (11/19)

4. two\_groups\_classfication\_multimodels (11/19)

尚不支持自动寻参的是：

1. model\_score\_with\_features\_add

2020/12/12: 所有本文件中函数

输入参数统一增加 savePath 用来存储图片

返回形式统一为：

df\_dict, str\_result, plot\_name\_list

分别为：dataframe从名字到内容的字典，字符串输出，图片文件名列表

2020/12/31:

1. features\_importance 入参增加 bool::standardization (数据标准化, default = True)

2. model\_score\_with\_features\_add 入参增加 bool::importance\_first (先进行XGBoost重要度排序，default = True)

3. two\_groups\_classfication\_multimodels 入参删除 4 个 (ylim\_min, ylim\_max, title1, title2)，后端views.py已修改

4. 所有图片保存前增填 savePath != None 的检查; savePath = None 作为内部调用使用，不额外保存图片

2021/01/13:

1. 所有函数增加变量筛选(调用 utils\_ml.ML\_assistant 中的 filtering 函数)

2021/02/09:

1. two\_class\_multimodel 森林图调用X5 forest\_plot()

2021/05/05:

前端修改要求：

feature\_importance 的 Advanced/views.py 中， model\_type 与方法的对应编号修改如下:

1 -- 回归/Lasso

2 -- 回归/岭回归

3 -- 回归/XGBoost

4 -- 回归/随机森林

5 -- 回归/AdaBoost

6 -- 分类/Logistic

7 -- 分类/XGBoost

8 -- 分类/随机森林

9 -- 分类/AdaBoost

Classification 增加一个 dataframe

"""

import os

import random

import datetime

import pickle

import math

import pandas as pd

import numpy as np

import matplotlib

matplotlib.use('AGG')

import matplotlib.pyplot as plt

from sklearn.base import clone

from sklearn.preprocessing import StandardScaler

from sklearn.inspection import permutation\_importance

from sklearn.model\_selection import cross\_val\_predict

from sklearn.model\_selection import cross\_validate

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

from sklearn.model\_selection import KFold

from sklearn.model\_selection import StratifiedKFold

from sklearn.model\_selection import ShuffleSplit

from sklearn.model\_selection import RandomizedSearchCV

from xgboost import XGBRegressor

from sklearn.linear\_model import LassoCV, RidgeCV

from sklearn.linear\_model import LogisticRegression, LogisticRegressionCV

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from sklearn.ensemble import AdaBoostRegressor

from sklearn.svm import LinearSVR

from sklearn.neighbors import KNeighborsRegressor

from sklearn.mixture import GaussianMixture

from sklearn.cluster import Birch

from sklearn.cluster import KMeans

from sklearn.cluster import AffinityPropagation

from sklearn.cluster import SpectralClustering

from sklearn.cluster import AgglomerativeClustering

# from lightgbm import LGBMClassifier

from xgboost import XGBClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import BaggingClassifier

from sklearn.ensemble import ExtraTreesClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.naive\_bayes import ComplementNB

from sklearn.svm import SVC

from sklearn.feature\_selection import RFECV

from sklearn.decomposition import PCA

from AnalysisFunction.utils\_ml.FeatrueSelect import mrmr\_classif

from AnalysisFunction.utils\_ml.FeatrueSelect import ReliefF

from sklearn.metrics import auc

from sklearn.metrics import silhouette\_score, calinski\_harabasz\_score, mutual\_info\_score, v\_measure\_score, \

normalized\_mutual\_info\_score

from sklearn.metrics import brier\_score\_loss

from sklearn.calibration import CalibratedClassifierCV, calibration\_curve

from sklearn.preprocessing import label\_binarize

from xgboost import plot\_importance

import AnalysisFunction.X\_5\_SmartPlot as x5

from AnalysisFunction.X\_5\_SmartPlot import plot\_calibration\_curve

from AnalysisFunction.X\_5\_SmartPlot import calculate\_net\_benefit

from AnalysisFunction.X\_5\_SmartPlot import plot\_decision\_curves

from AnalysisFunction.X\_1\_DataGovernance import data\_standardization

from AnalysisFunction.X\_1\_DataGovernance import \_analysis\_dict

from AnalysisFunction.X\_2\_DataSmartStatistics import comprehensive\_smart\_analysis

from AnalysisFunction.utils\_ml import filtering, dic2str, round\_dec, save\_fig

from AnalysisFunction.utils\_ml import classification\_metric\_evaluate, regression\_metric\_evaluate

from AnalysisFunction.utils\_ml import make\_class\_metrics\_dict, make\_regr\_metrics\_dict, multiclass\_metric\_evaluate

from AnalysisFunction.utils\_ml import ci

from AnalysisFunction.utils\_ml import GridSearcherCV, RandSearcherCV, GridSearcherSelf, RandSearcherSelf

from AnalysisFunction.utils\_ml.params import RandDefaultRange

from AnalysisFunction.utils\_ml.auc\_delong import delong\_roc\_test

from functools import reduce

plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签

plt.rcParams['axes.unicode\_minus'] = False # 用来正常显示负号)

from matplotlib import rc

plt.rcParams['ps.useafm'] = True

rc('font', \*\*{'family': 'sans-serif', 'sans-serif': ['FreeSans']})

plt.rcParams['pdf.fonttype'] = 42

# -------------------------------------------------------------

# ----------------------分类多模型综合分析------------------------

# -------------------------------------------------------------

def two\_groups\_classfication\_multimodels(

df\_input,

group,

features,

methods=[],

decimal\_num=3,

testsize=0.2,

boostrap=5,

randomState=42,

smooth=False,

searching=False,

dpi=600,

picFormat='jpeg',

isKFold=True,

savePath=None,

resultType=0,

delong=False,

\*\*kwargs,

):

"""

df\_input:Dataframe

features:自变量list

group：因变量str

testsize: 测试集比例

boostrap：重采样次数

searching:bool 是否进行自动寻参，默认为否

savePath:str 图片存储路径

"""

str\_time = str(datetime.datetime.now().hour) + str(datetime.datetime.now().minute) + str(

datetime.datetime.now().second)

random\_number = random.randint(1, 100)

str\_time = str\_time + str(random\_number)

dftemp = df\_input[features + [group]].dropna()

features\_flag = False

x = dftemp[features]

y = dftemp[[group]]

u = np.sort(np.unique(np.array(dftemp[group])))

if len(u) == 2 and set(u) != set([0, 1]):

y\_result = label\_binarize(dftemp[group], classes=[ii for ii in u]) # 将标签二值化

y\_result\_pd = pd.DataFrame(y\_result, columns=[group])

df = pd.concat([dftemp.drop(group, axis=1), y\_result\_pd], axis=1)

x = df[features]

y = df[[group]]

elif len(u) > 2:

return {'error': '暂时只支持二分类。请检查因变量取值情况。'}

name\_dict = {

'LogisticRegression': 'logistic',

'XGBClassifier': 'XGBoost',

'RandomForestClassifier': 'RandomForest',

'LGBMClassifier': 'LightGBM',

'SVC': 'SVM',

'MLPClassifier': 'MLP',

'GaussianNB': 'GNB',

'ComplementNB': 'CNB',

'AdaBoostClassifier': 'AdaBoost',

'KNeighborsClassifier': 'KNN',

'DecisionTreeClassifier': 'DecisionTree',

'BaggingClassifier': 'Bagging',

}

if len(methods) == 0:

methods = [

'LogisticRegression',

'XGBClassifier',

'RandomForestClassifier',

# 'SVC',

# 'MLPClassifier',

# 'AdaBoostClassifier',

# 'KNeighborsClassifier',

# 'DecisionTreeClassifier',

# 'BaggingClassifier',

]

str\_result = '已采用多种机器学习模型尝试完成数据样本分类任务，包括：{}。各模型的参数值选取情况如下所示：\n\n'.format(methods)

plot\_name\_list = []

plot\_name\_dict\_save = {}

plot\_name\_dict ={}

fig, ax = plt.subplots(figsize=(4, 4), dpi=dpi)

# 画对角线

ax.plot(

[0, 1], [0, 1],

linestyle='--',

lw=1, color='r',

alpha=1.0,

)

ax.grid(which='major', axis='both', linestyle='-.', alpha=0.3, color='grey')

ax.set\_xlim([-0.02, 1.02])

ax.set\_ylim([-0.02, 1.02])

ax.tick\_params(top=False, right=False)

# ax.legend(bbox\_to\_anchor=(1.05, 1), loc=2, borderaxespad=0)

ax.set\_xlabel('1-Specificity')

ax.set\_ylabel('Sensitivity')

ax.set\_title('Validation ROC Curve')

mean\_fpr = np.linspace(0, 1, 100)

colors = x5.CB91\_Grad\_BP

df\_0 = pd.DataFrame(columns=list(make\_class\_metrics\_dict().keys()), index=[0])

df\_0\_test = df\_0.copy()

df\_plot = pd.DataFrame(columns=['method', 'mean', 'std'])

fpr\_train\_alls, tpr\_train\_alls, train\_method\_alls, mean\_auc\_train\_alls = [], [], [], []

fraction\_of\_positives\_alls, mean\_predicted\_value\_alls, clf\_score\_alls = [], [], []

AUC\_95CI\_test, AUC\_95CI\_SD\_test, AUC\_95CI\_train, AUC\_95CI\_SD\_train = [], [], [], []

DCA\_dict = {}

model\_test\_data\_all = {}

X\_train\_ps, Y\_train\_ps, model\_train\_s = [], [], []

X\_test\_ps, Y\_test\_ps = [], []

name = []

for i, method in enumerate(methods):

tprs\_train, tprs\_test = [], []

name.append(name\_dict[method])

if searching == True:

if method == 'LGBMClassifier':

searcher = GridSearcherCV('Classification', globals()[method]())

selected\_model = searcher(x, y)

else:

searcher = RandSearcherCV('Classification', globals()[method]())

selected\_model = searcher(x, y) # ; searcher.report()

elif searching == False:

# selected\_model = globals()[method]() if (method != 'SVC') else globals()[method](probability=True)

if method == 'SVC':

selected\_model = globals()[method](probability=True)

elif method == 'MLPClassifier':

selected\_model = globals()[method](hidden\_layer\_sizes=(20, 10), max\_iter=20)

elif method == 'RandomForestClassifier':

selected\_model = globals()[method](n\_estimators=20)

else:

selected\_model = globals()[method]()

elif searching == 'Handle':

method\_dicts = kwargs

if i == 0:

me\_count = True

for me\_list in methods:

if me\_list in method\_dicts.keys():

me\_count = False

continue

if me\_count:

return {'error': '请设置要调参的模型！'}

if method in method\_dicts.keys():

method\_dict = {}

if (method == 'SVC'):

method\_dict.update({'probability': True})

method\_dict.update(method\_dicts[method])

if (method == 'RandomForestClassifier' and method\_dict['max\_depth'] == 'None'):

method\_dict['max\_depth'] = None

if (method == 'MLPClassifier'):

hls\_vals = str(method\_dict['hidden\_layer\_sizes']).split(',')

hls\_value = ()

for hls\_val in hls\_vals:

try:

if int(hls\_val) >= 5 and int(hls\_val) <= 200:

hls\_value = hls\_value + (int(hls\_val),)

else:

return {'error': '请按照要求重新设置隐藏层宽度！'}

except:

return {'error': '请重新设神经网络模型中的隐藏层宽度！'}

method\_dict['hidden\_layer\_sizes'] = hls\_value

if (method == 'GaussianNB' and method\_dict['priors'] == 'None'):

method\_dict['priors'] = None

elif (method == 'GaussianNB'):

pri\_vals = str(method\_dict['priors']).split(',')

pri\_value = ()

pri\_sum = 0.0

for pri\_val in pri\_vals:

try:

pri\_sum = float(pri\_val) + pri\_sum

pri\_value = pri\_value + (float(pri\_val),)

except:

return {'error': '请重新设朴素贝叶斯模型中的先验概率！'}

if len(pri\_vals) == len(y.unique()) and pri\_sum == 1.0:

method\_dict['priors'] = pri\_value

else:

return {'error': '请重新设朴素贝叶斯模型中的先验概率！'}

selected\_model = globals()[method](\*\*method\_dict)

else:

if method == 'LGBMClassifier':

searcher = GridSearcherCV('Classification', globals()[method]())

selected\_model = searcher(x, y)

else:

searcher = RandSearcherCV('Classification', globals()[method]())

selected\_model = searcher(x, y) # ; searcher.report()

list\_evaluate\_dic\_train = make\_class\_metrics\_dict()

list\_evaluate\_dic\_test = make\_class\_metrics\_dict()

clf\_score = 1

fraction\_of\_positives = np.array([1])

mean\_predicted\_value = np.array([1])

p\_serie\_s\_te, net\_benefit\_serie\_s\_te, net\_benefit\_serie\_All\_s\_te = [], [], []

data\_all = {}

test\_data\_delong = {}

conf\_dic\_train, conf\_dic\_test = {}, {}

if isKFold:

# KF = KFold(n\_splits=boostrap, random\_state=42,shuffle=True)

KF = StratifiedKFold(n\_splits=boostrap, random\_state=randomState, shuffle=True)

for i\_k, (train\_index, valid\_index) in enumerate(KF.split(x, y)):

# 划分训练集和验证集

Xtrain, Xtest = x.iloc[train\_index], x.iloc[valid\_index]

Ytrain, Ytest = y.iloc[train\_index], y.iloc[valid\_index]

data\_all.update({i\_k: {'Xtrain': Xtrain, 'Ytrain': Ytrain, 'Xtest': Xtest, 'Ytest': Ytest}})

test\_data\_delong.update({i\_k: np.array(Ytest).T[0]})

else:

for index in range(0, boostrap):

if searching == 'Handle':

Xtrain, Xtest, Ytrain, Ytest = TTS(x, y, test\_size=testsize, random\_state=index)

else:

Xtrain, Xtest, Ytrain, Ytest = TTS(x, y, test\_size=testsize)

data\_all.update({index: {'Xtrain': Xtrain, 'Ytrain': Ytrain, 'Xtest': Xtest, 'Ytest': Ytest}})

test\_data\_delong.update({index: np.array(Ytest).T[0]})

if method == methods[0]:

model\_test\_data\_all.update({'original': test\_data\_delong})

# for index in range(0, boostrap):

test\_all\_data\_delong = {}

X\_train\_p, Y\_train\_p, model\_train = [], [], []

X\_test\_p, Y\_test\_p = [], []

for data\_key, data\_value in data\_all.items():

Xtrain, Ytrain, Xtest, Ytest = data\_value['Xtrain'], data\_value['Ytrain'], data\_value['Xtest'], data\_value[

'Ytest']

model = clone(selected\_model).fit(Xtrain, Ytrain)

####################################

# if data\_key == 0:

X\_train\_p.append(Xtrain)

Y\_train\_p.append(Ytrain)

model\_train.append(model)

X\_test\_p.append(Xtest)

Y\_test\_p.append(Ytest)

##########################################

Yprob = model.predict\_proba(Xtest)[:, 1]

test\_all\_data\_delong.update({data\_key: Yprob})

prob\_pos, p\_serie, net\_benefit\_serie, net\_benefit\_serie\_All = calculate\_net\_benefit(model, Xtest, Ytest)

p\_serie\_s\_te.append(p\_serie)

net\_benefit\_serie\_s\_te.append(net\_benefit\_serie)

net\_benefit\_serie\_All\_s\_te.append(net\_benefit\_serie\_All)

"""

if hasattr(model, "predict\_proba"):

prob\_pos = model.predict\_proba(Xtest)[:, 1]

else: # use decision function

prob\_pos = model.decision\_function(Xtest)

prob\_pos = (prob\_pos - prob\_pos.min()) / (prob\_pos.max() - prob\_pos.min())

"""

clf\_score1 = brier\_score\_loss(Ytest, prob\_pos, pos\_label=y[group].max()) ##strategy='quantile',

if clf\_score > clf\_score1:

clf\_score = clf\_score1

fraction\_of\_positives, mean\_predicted\_value = calibration\_curve(Ytest, prob\_pos,

n\_bins=10)

# 利用classification\_metric\_evaluate函数获取在测试集的预测值

try:

fpr\_train, tpr\_train, metric\_dic\_train, \_ = classification\_metric\_evaluate(model, Xtrain, Ytrain)

fpr\_test, tpr\_test, metric\_dic\_test, \_ = classification\_metric\_evaluate(model, Xtest, Ytest,

Threshold=metric\_dic\_train[

'cutoff'])

metric\_dic\_test.update({'cutoff': metric\_dic\_train['cutoff']})

except Exception as e:

return {'error': '数据不均衡，至少有一组验证集中存在结局全部为0或者1的数据！请选择另外一种方法重采样（交叉验证）的方法处理！'}

# interp:插值 把结果添加到tprs列表中

tprs\_train.append(np.interp(mean\_fpr, fpr\_train, tpr\_train))

tprs\_test.append(np.interp(mean\_fpr, fpr\_test, tpr\_test))

tprs\_train[-1][0] = 0.0

tprs\_test[-1][0] = 0.0

# 计算所有评价指标

for key in list\_evaluate\_dic\_train.keys():

list\_evaluate\_dic\_train[key].append(metric\_dic\_train[key])

list\_evaluate\_dic\_test[key].append(metric\_dic\_test[key])

model\_test\_data\_all.update({method: test\_all\_data\_delong})

DCA\_dict[name\_dict[method]] = {'p\_serie': p\_serie\_s\_te, 'net\_b\_s': net\_benefit\_serie\_s\_te,

'net\_b\_s\_A': net\_benefit\_serie\_All\_s\_te}

X\_train\_ps.append(X\_train\_p)

Y\_train\_ps.append(Y\_train\_p)

model\_train\_s.append(model\_train)

X\_test\_ps.append(X\_test\_p)

Y\_test\_ps.append(Y\_test\_p)

###画校准曲线

# X\_train, X\_test, Y\_train, Y\_test = TTS(x, y, test\_size=testsize, random\_state=0)

# model\_CC = clone(selected\_model).fit(X\_train, Y\_train)

# y\_pred = model.predict(Xtest)

# if hasattr(model\_CC, "predict\_proba"):

# prob\_pos = model\_CC.predict\_proba(X\_test)[:, 1]

# else: # use decision function

# prob\_pos = model\_CC.decision\_function(X\_test)

# prob\_pos = (prob\_pos - prob\_pos.min()) / (prob\_pos.max() - prob\_pos.min())

# clf\_score = brier\_score\_loss(Y\_test, prob\_pos, pos\_label=y.max())##strategy='quantile',

# fraction\_of\_positives, mean\_predicted\_value = calibration\_curve(Y\_test, prob\_pos,

# n\_bins=10)

clf\_score\_alls.append(clf\_score)

fraction\_of\_positives\_alls.append(fraction\_of\_positives)

mean\_predicted\_value\_alls.append(mean\_predicted\_value)

for key in list\_evaluate\_dic\_train.keys():

metric\_dic\_train[key] = np.mean(list\_evaluate\_dic\_train[key])

metric\_dic\_test[key] = np.mean(list\_evaluate\_dic\_test[key])

if resultType == 0: ##SD

list\_evaluate\_dic\_train[key] = np.std(list\_evaluate\_dic\_train[key], axis=0)

list\_evaluate\_dic\_test[key] = np.std(list\_evaluate\_dic\_test[key], axis=0)

elif resultType == 1: ##CI

conf\_dic\_train[key] = list(ci(list\_evaluate\_dic\_train[key]))

conf\_dic\_test[key] = list(ci(list\_evaluate\_dic\_test[key]))

result\_dic\_train = metric\_dic\_train

result\_dic\_test = metric\_dic\_test

if resultType == 0: ##SD

for tem in ['AUC\_L', 'AUC\_U']:

del list\_evaluate\_dic\_train[tem]

del list\_evaluate\_dic\_test[tem]

for key in list\_evaluate\_dic\_train.keys():

if key == 'AUC':

result\_dic\_train['AUC(95%CI)'] = str(round\_dec(float(metric\_dic\_train[key]), d=decimal\_num)) + '(' + \

str(round\_dec(float(list\_evaluate\_dic\_train[key]),

d=decimal\_num)) + ')'

result\_dic\_test['AUC(95%CI)'] = str(round\_dec(float(metric\_dic\_test[key]), d=decimal\_num)) + '(' + \

str(round\_dec(float(list\_evaluate\_dic\_test[key]),

d=decimal\_num)) + ')'

else:

result\_dic\_train[key] = str(round\_dec(float(metric\_dic\_train[key]), d=decimal\_num)) + '(' + \

str(round\_dec(float(list\_evaluate\_dic\_train[key]), d=decimal\_num)) + ')'

result\_dic\_test[key] = str(round\_dec(float(metric\_dic\_test[key]), d=decimal\_num)) + '(' + \

str(round\_dec(float(list\_evaluate\_dic\_test[key]), d=decimal\_num)) + ')'

elif resultType == 1:

for tem in ['AUC\_L', 'AUC\_U']:

del conf\_dic\_train[tem]

del conf\_dic\_test[tem]

for key in conf\_dic\_train.keys():

if key == 'AUC':

result\_dic\_train['AUC(95%CI)'] = str(round\_dec(float(metric\_dic\_train[key]), decimal\_num)) + ' (' + \

str(round\_dec(float(metric\_dic\_train['AUC\_L']),

decimal\_num)) + '-' + \

str(round\_dec(float(metric\_dic\_train['AUC\_U']), decimal\_num)) + '）'

result\_dic\_test['AUC(95%CI)'] = str(round\_dec(float(metric\_dic\_test[key]), decimal\_num)) + ' (' + \

str(round\_dec(float(metric\_dic\_test['AUC\_L']), decimal\_num)) + '-' + \

str(round\_dec(float(metric\_dic\_test['AUC\_U']), decimal\_num)) + '）'

else:

result\_dic\_train[key] = str(round\_dec(float(metric\_dic\_train[key]), d=decimal\_num)) + '(' + \

str(round\_dec(float(conf\_dic\_train[key][0]), d=decimal\_num)) + '-' + \

str(round\_dec(float(conf\_dic\_train[key][1]), d=decimal\_num)) + ')'

result\_dic\_test[key] = str(round\_dec(float(metric\_dic\_test[key]), d=decimal\_num)) + '(' + \

str(round\_dec(float(conf\_dic\_test[key][0]), d=decimal\_num)) + '-' + \

str(round\_dec(float(conf\_dic\_test[key][1]), d=decimal\_num)) + ')'

df\_train\_result = pd.DataFrame([result\_dic\_train], index=['Mean'])

df\_test\_result = pd.DataFrame([result\_dic\_test], index=['Mean'])

df\_train\_result['分类模型'] = name\_dict[method]

df\_test\_result['分类模型'] = name\_dict[method]

AUC\_95CI\_test.append(list(df\_test\_result.iloc[0, -4:-2]))

AUC\_95CI\_train.append(list(df\_train\_result.iloc[0, -4:-2]))

df\_0 = pd.concat([df\_0, df\_train\_result])

df\_0\_test = pd.concat([df\_0\_test, df\_test\_result])

mean\_tpr\_train = np.mean(tprs\_train, axis=0)

mean\_tpr\_test = np.mean(tprs\_test, axis=0)

mean\_tpr\_train[-1] = 1.0

mean\_tpr\_test[-1] = 1.0

mean\_auc\_train = auc(mean\_fpr, mean\_tpr\_train) # 计算训练集平均AUC值

mean\_auc\_test = auc(mean\_fpr, mean\_tpr\_test)

###画训练集ROC

fpr\_train\_alls.append(mean\_fpr)

tpr\_train\_alls.append(mean\_tpr\_train)

train\_method\_alls.append(method)

mean\_auc\_train\_alls.append(mean\_auc\_train)

std\_value = 0

if resultType == 0:

std\_value = list\_evaluate\_dic\_test['AUC']

elif resultType == 1:

std\_value = (conf\_dic\_train['AUC'][1] - conf\_dic\_train['AUC'][0]) / 2

df\_plot = df\_plot.append({

'method': name\_dict[method],

'mean': mean\_auc\_test,

'std': std\_value,

}, ignore\_index=True)

ax.plot(mean\_fpr, mean\_tpr\_test, c=colors[i], label=name\_dict[method] + '(AUC = %0.3f 95%%CI (%0.3f-%0.3f))' % (

df\_test\_result.iloc[0, 0], df\_test\_result.iloc[0, -4], df\_test\_result.iloc[0, -3]),

lw=1.5, alpha=1)

str\_result += method + ': AUC=' + str(round\_dec(mean\_auc\_train, decimal\_num)) + '; 模型参数:\n' + dic2str(

selected\_model.get\_params(),

method) + '\n'

###模型德龙检测

if delong:

delong\_z, delong\_p = [], []

for i in range(boostrap):

zzz, ppp = [], []

for method1 in methods:

zz, pp = [], []

for method2 in methods:

z, p = delong\_roc\_test(model\_test\_data\_all['original'][i], model\_test\_data\_all[method1][i],

model\_test\_data\_all[method2][i])

zz.append(z[0][0])

pp.append(p[0][0])

zzz.append(zz)

ppp.append(pp)

delong\_z.append(zzz)

delong\_p.append(ppp)

if boostrap == 1:

delong\_zz1 = pd.DataFrame(reduce(lambda x, y: np.array(x) + np.array(y), delong\_z),

index=methods,

columns=methods)

delong\_pp1 = pd.DataFrame(reduce(lambda x, y: np.array(x) + np.array(y), delong\_p),

index=methods,

columns=methods)

else:

delong\_zz1 = pd.DataFrame(reduce(lambda x, y: np.array(x) + np.array(y), delong\_z) / len(delong\_z),

index=methods,

columns=methods)

delong\_pp1 = pd.DataFrame(reduce(lambda x, y: np.array(x) + np.array(y), delong\_p) / len(delong\_p),

index=methods,

columns=methods)

delong\_zz = delong\_zz1.applymap(lambda x: round\_dec(x, d=decimal\_num))

delong\_pp = delong\_pp1.applymap(lambda x: round\_dec(x, d=decimal\_num))

# ymin = min([y - dy for y, dy in zip(df\_plot['mean'], df\_plot['std'])])

# ymax = max([y + dy for y, dy in zip(df\_plot['mean'], df\_plot['std'])])

# ymin, ymax = ymin - (ymax - ymin) / 4.0, ymax + (ymax - ymin) / 10.0

if boostrap != 1:

ymax = np.max(df\_plot['mean']) + np.max(df\_plot['std']) + (np.max(df\_plot['mean']) - np.min(df\_plot['mean'])) / 4

ymin = np.min(df\_plot['mean']) - np.max(df\_plot['std']) - (np.max(df\_plot['mean']) - np.min(df\_plot['mean'])) / 4

ymax = math.ceil(ymax \* 100) / 100

ymin = int(ymin \* 100) / 100

ax.legend(loc="lower right", fontsize=5)

ax.legend(loc="lower right", fontsize=5)

df\_test\_auc = []

if savePath is not None:

plot\_name\_list.append(save\_fig(savePath, 'valid\_ROC\_curve', 'png', fig, str\_time=str\_time))

plot\_name\_dict\_save['验证集ROC曲线'] = save\_fig(savePath, 'valid\_ROC\_curve', picFormat, fig, str\_time=str\_time)

# 画训练集ROC

fig1 = plt.figure(figsize=(4, 4), dpi=dpi)

# 画对角线

plt.plot(

[0, 1], [0, 1],

linestyle='--',

lw=1, color='r',

alpha=0.8,

)

plt.grid(which='major', axis='both', linestyle='-.', alpha=0.3, color='grey')

for i in range(len(fpr\_train\_alls)):

df\_test\_auc.append(df\_0.iloc[i + 1]['AUC'])

plt.plot(

fpr\_train\_alls[i], tpr\_train\_alls[i],

lw=1.5, alpha=0.9,

c=colors[i],

label=name\_dict[train\_method\_alls[i]] + '(AUC = %0.3f 95%%CI (%0.3f-%0.3f))' % (

df\_0.iloc[i + 1]['AUC'], AUC\_95CI\_train[i][0], AUC\_95CI\_train[i][1])

)

plt.xlim([-0.02, 1.02])

plt.ylim([-0.02, 1.02])

plt.xlabel('1-Specificity')

plt.ylabel('Sensitivity')

plt.title('Train ROC Curve')

plt.legend(loc='lower right', fontsize=5)

plot\_name\_list.append(save\_fig(savePath, 'ROC\_Train\_curve', 'png', fig1, str\_time=str\_time))

plot\_name\_dict\_save['训练集ROC曲线图'] = save\_fig(savePath, 'ROC\_Train\_curve', picFormat, fig1, str\_time=str\_time)

plot\_name\_list.reverse() ###所有图片倒置

if boostrap != 1:

# df\_plot.drop('mean', axis=1)

# df\_plot.loc[:,'mean']=pd.Series(df\_test\_auc,name='mean')

plot\_name\_list += x5.forest\_plot(

df\_input=df\_plot,

name='method', value='mean', err='std', direct='horizontal',

fig\_size=[len(methods) + 3, 9],

ylim=[ymin, ymax],

title='Forest Plot of Each Model AUC Score ',

path=savePath,

dpi=dpi,

picFormat=picFormat,

)

plot\_name\_dict\_save['验证集多模型森林图'] = plot\_name\_list[len(plot\_name\_list) - 1]

plot\_name\_list.pop(len(plot\_name\_list) - 1)

plt.close()

###画校准曲线

if savePath is not None:

from scipy.optimize import curve\_fit

from scipy.interpolate import make\_interp\_spline

def fit\_f(x, a, b):

return a \* np.arcsin(x) + b

def fit\_show(x, y\_fit):

a, b = y\_fit.tolist()

return a \* np.arcsin(x) + b

fig, ax1 = plt.subplots(figsize=(6, 6), dpi=dpi)

ax1.plot([0, 1], [0, 1], "k:", label="Perfectly Calibrated")

for i in range(len(mean\_predicted\_value\_alls)):

if smooth and len(fraction\_of\_positives\_alls[i]) >= 3:

x\_new = np.linspace(min(mean\_predicted\_value\_alls[i]), max(mean\_predicted\_value\_alls[i]),

len(fraction\_of\_positives\_alls[i]) \* 10)

try:

p\_fit, \_ = curve\_fit(fit\_f, mean\_predicted\_value\_alls[i], fraction\_of\_positives\_alls[i],

maxfev=10000)

y\_smooth = fit\_show(x\_new, p\_fit)

# y\_fit = np.polyfit(mean\_predicted\_value\_alls[i], fraction\_of\_positives\_alls[i], 3)

# y\_smooth = f\_fit(x\_new, y\_fit)

# y\_smooth = spline(mean\_predicted\_value\_alls[i], fraction\_of\_positives\_alls[i], x\_new)

ax1.plot(x\_new, y\_smooth, c=colors[i],

label="%s (%1.3f)" % (name\_dict[methods[i]], clf\_score\_alls[i]))

except Exception as e:

ax1.plot(mean\_predicted\_value\_alls[i], fraction\_of\_positives\_alls[i], "s-", c=colors[i],

label="%s (%1.3f)" % (name\_dict[methods[i]], clf\_score\_alls[i]))

else:

ax1.plot(mean\_predicted\_value\_alls[i], fraction\_of\_positives\_alls[i], "s-", c=colors[i],

label="%s (%1.3f)" % (name\_dict[methods[i]], clf\_score\_alls[i]))

ax1.set\_xlabel("Mean predicted value")

ax1.set\_ylabel("Fraction of positives")

ax1.set\_ylim([-0.05, 1.05])

ax1.legend(loc="lower right")

ax1.set\_title('Calibration plots (reliability curve)')

plt.gca()

plt.close()

plot\_name = "Calibration\_curve\_" + str\_time

plot\_name\_list.append(save\_fig(savePath, plot\_name, 'png', fig, str\_time=str\_time))

plot\_name\_dict\_save['验证集多模型校准曲线'] = save\_fig(savePath, plot\_name, picFormat, fig, str\_time=str\_time)

###画DCA曲线

if savePath is not None:

decision\_curve\_p = plot\_decision\_curves(DCA\_dict, colors=colors, name='Valid', savePath=savePath, dpi=dpi,

picFormat=picFormat)

plot\_name\_list.append(decision\_curve\_p[0])

plot\_name\_dict\_save['验证集DCA曲线图'] = decision\_curve\_p[1]

###画PR曲线

if savePath is not None:

# from sklearn.metrics import plot\_precision\_recall\_curve

from AnalysisFunction.X\_5\_SmartPlot import plot\_precision\_recall\_curve

fig = plot\_precision\_recall\_curve(model\_train\_s, X\_train\_ps, Y\_train\_ps, name=name, picname='train')

plot\_name\_dict['训练集多模型PR曲线'] = save\_fig(savePath, 'PR\_train', 'png', fig, str\_time=str\_time)

plot\_name\_dict\_save['训练集多模型PR曲线'] = save\_fig(savePath, 'PR\_train', picFormat, fig, str\_time=str\_time)

plt.close(fig)

fig =plot\_precision\_recall\_curve(model\_train\_s, X\_test\_ps, Y\_test\_ps, name=name, picname='Validation')

plot\_name\_dict['验证集多模型PR曲线'] = save\_fig(savePath, 'PR\_valid', 'png', fig, str\_time=str\_time)

plot\_name\_dict\_save['验证集多模型PR曲线'] = save\_fig(savePath, 'PR\_vlid', picFormat, fig, str\_time=str\_time)

plt.close(fig)

df\_train\_result1 = df\_0.drop([0])

df\_test\_result1 = df\_0\_test.drop([0])

classfier = df\_train\_result1.pop('分类模型')

df\_train\_result = df\_train\_result1.applymap(lambda x: round\_dec(x, d=decimal\_num))

df\_train\_result.insert(0, '分类模型', classfier)

df\_test\_result1.pop('分类模型')

df\_test\_result = df\_test\_result1.applymap(lambda x: round\_dec(x, d=decimal\_num))

df\_test\_result.insert(0, '分类模型', classfier)

AUC\_95CI\_tr = df\_train\_result.pop('AUC(95%CI)')

df\_train\_result.insert(1, 'AUC(95%CI)', AUC\_95CI\_tr)

AUC\_95CI\_te = df\_test\_result.pop('AUC(95%CI)')

df\_test\_result.insert(1, 'AUC(95%CI)', AUC\_95CI\_te)

df\_train\_result = df\_train\_result.drop(['AUC\_L', 'AUC\_U'], axis=1)

df\_test\_result = df\_test\_result.drop(['AUC\_L', 'AUC\_U'], axis=1)

if features\_flag:

df\_count\_r = round\_dec(Xtest.shape[0] / x.shape[0], decimal\_num)

else:

df\_count\_r = round\_dec(testsize, decimal\_num)

if isKFold:

str\_result += '\n下示森林图展示了各模型进行' + group + '预测的ROC结果,图中的误差线为ROC均值及SD。\n' \

+ '模型的ROC均值及SD的是通过' + str(boostrap) + '折交叉验证,' + '模型中的变量包括' \

+ ','.join(features) + '。\n'

else:

str\_result += '\n下示森林图展示了各模型进行' + group + '预测的ROC结果,图中的误差线为ROC均值及SD。\n' \

+ '模型的ROC均值及SD的是通过多次重复采样计算，重复采样次数为' + str(boostrap) + '次,' \

+ '每一次重采样训练的验证集占总体样本的' + str(df\_count\_r \* 100) + '%,训练集占' \

+ str((1 - df\_count\_r) \* 100) + '%,' + '模型中的变量包括' \

+ ','.join(features) + '。\n'

best\_ = df\_train\_result.loc[df\_train\_result.index == 'Mean'].sort\_values(by='AUC', ascending=False).head(1)

name\_train = best\_.iloc[0]['分类模型']

str\_result += '在目前所有模型中，训练集表现最佳者为{}（依据AUC排序），在各评价标准中其在训练集对应分数分别为：\n'.format(name\_train)

for col in best\_.columns[1:]:

str\_result += '\t{}：{}\n'.format(col, best\_.iloc[0][col])

best\_ = df\_test\_result.loc[df\_test\_result.index == 'Mean'].sort\_values(by='AUC', ascending=False).head(1)

name\_test = best\_.iloc[0]['分类模型']

str\_result += '验证集表现最佳者为{}（依据AUC排序），在各评价标准中其在验证集对应分数分别为：\n'.format(name\_test)

for col in best\_.columns[1:]:

str\_result += '\t{}：{}\n'.format(col, best\_.iloc[0][col])

if (name\_test == name\_train):

str\_result += '二者吻合，可以认为{}是针对此数据集的最佳模型选择。'.format(name\_train)

else:

str\_result += '二者不吻合，{}极可能存在过拟合现象，{}可能稳定性相对较好。具体模型选择可根据下表详细评分信息进行取舍。'.format(name\_train, name\_test)

if resultType == 0:

df\_train\_result.rename(columns={"AUC(95%CI)": 'AUC(SD)', 'cutoff': 'cutoff(SD)', '准确度': '准确度(SD)',

'灵敏度': '灵敏度(SD)', '特异度': '特异度(SD)',

'阳性预测值': '阳性预测值(SD)', '阴性预测值': '阴性预测值(SD)',

'F1分数': 'F1分数(SD)', 'Kappa': 'Kappa(SD)'}, inplace=True)

df\_test\_result.rename(columns={"AUC(95%CI)": 'AUC(SD)', 'cutoff': 'cutoff(SD)', '准确度': '准确度(SD)',

'灵敏度': '灵敏度(SD)', '特异度': '特异度(SD)',

'阳性预测值': '阳性预测值(SD)', '阴性预测值': '阴性预测值(SD)',

'F1分数': 'F1分数(SD)', 'Kappa': 'Kappa(SD)'}, inplace=True)

elif resultType == 1:

df\_train\_result.rename(columns={'cutoff': 'cutoff(95%CI)', '准确度': '准确度(95%CI)',

'灵敏度': '灵敏度(95%CI)', '特异度': '特异度(95%CI)',

'阳性预测值': '阳性预测值(95%CI)', '阴性预测值': '阴性预测值(95%CI)',

'F1分数': 'F1分数(95%CI)', 'Kappa': 'Kappa(95%CI)'}, inplace=True)

df\_test\_result.rename(columns={'cutoff': 'cutoff(95%CI)', '准确度': '准确度(95%CI)',

'灵敏度': '灵敏度(95%CI)', '特异度': '特异度(95%CI)',

'阳性预测值': '阳性预测值(95%CI)', '阴性预测值': '阴性预测值(95%CI)',

'F1分数': 'F1分数(95%CI)', 'Kappa': 'Kappa(95%CI)'}, inplace=True)

df\_dict = {

'多模型分类-训练集结果汇总': df\_train\_result.drop(['AUC'], axis=1),

'多模型分类-验证集结果汇总': df\_test\_result.drop(['AUC'], axis=1),

}

if delong:

df\_dict.update({'delong检测Z值均值表': delong\_zz.fillna('NaN', inplace=True)})

df\_dict.update({'delong检测P值均值表': delong\_pp.fillna('NaN', inplace=True)})

if boostrap != 1:

plot\_name\_dict = {

'训练集ROC曲线图': plot\_name\_list[0],

'验证集ROC曲线图': plot\_name\_list[1],

'验证集多模型森林图': plot\_name\_list[2],

'验证集多模型校准曲线': plot\_name\_list[3],

'验证集DCA曲线图': plot\_name\_list[4],

}

else:

plot\_name\_dict = {

'训练集ROC曲线图': plot\_name\_list[0],

'验证集ROC曲线图': plot\_name\_list[1],

'验证集多模型校准曲线': plot\_name\_list[2],

'验证集DCA曲线图': plot\_name\_list[3],

}

result\_dict = {'str\_result': {'分析结果描述': str\_result}, 'tables': df\_dict,

'pics': plot\_name\_dict, 'save\_pics': plot\_name\_dict\_save}

return result\_dict

def DataSplit(df, label='LABEL', validationRatio=0.15, randomState=1, decimal\_num=3):

"""

数据拆分

:param df: dataframe 数据

:param label: str 标签列名称

:param validationRatio:测试集比例

:param randomState:随机种子

:param decimal\_num:小数点位数

:return:

"""

features = df.columns.values.tolist()

if label in features: ##label中0为训练集，1为测试集#0.005,

df = df.drop([label], axis=1)

train, test = TTS(df, test\_size=validationRatio, random\_state=randomState, )

train[label] = list(map(lambda x: int(x), np.zeros(len(train))))

test[label] = list(map(lambda x: int(x), np.ones(len(test))))

result\_df = pd.concat([train, test], axis=0)

# result\_df[label] = pd.DataFrame(list(map(lambda x: int(x), result\_df[label])))

str\_result = '数据划分标签列为(0为训练集，1为测试集)：' + label + '\n' + '其中训练集共有样本数：' + str(len(train)) + '\n' + '其中测试集共有样本数：' + str(

len(test))

result\_dict = {'str\_result': str\_result,

'tables': {'存储数据表': result\_df},

'pics': None}

return result\_dict

#!/usr/bin/python3

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

"""

2020/11/19 by Owen Yang

##hz 修改机器学习验证集和测试集结果一致问题 ML\_Classfication

外部可引用函数:

特征分析选择:

func::principle\_component\_analysis:

func::feature importance:

regression based: Lasso, Ridge

classification based: xgboost(classifier)

func::model\_score\_with\_features\_add:

classification based: xgboost classifier, Logistic regression, SVC,

数据建模:

func::ML\_Classification:

'LogisticRegression',

'XGBClassifier',

'RandomForestClassifier',

'SVC',

'KNeighborsClassifier',

func::ML\_Regression:

'LinearRegression',

'XGBRegressor',

'RandomForestRegressor',

'LinearSVR',

'KNeighborsRegressor',

func::ML\_Clustering:

'KMeans',

'Birch',

'SpectralClustering',

'AgglomerativeClustering',

'GaussianMixture',

模型对比:

funct::two\_groups\_classfication\_multimodels:

'XGBClassifier'

'LogisticRegression',

'SVC',

'MLPClassifier',

'RandomForestClassifier',

'AdaBoostClassifier',

# 'KNeighborsClassifier',

# 'DecisionTreeClassifier',

# 'GradientBoostingClassifier',

# 'BaggingClassifier',

# 'ExtraTreesClassifier',

内部函数:

供 func::feature importance 使用:

func::\_lasso\_features\_importance

func::\_ridge\_features\_importance

func::\_xgboost\_features\_importance

自动寻参:

对于监督式学习(分类和回归)：

GridSearcherCV: 利用交叉验证为选择标准，对所有可能参数构型进行遍历搜索

RandSearcherCV: 利用交叉验证为选择标准，对所有可能参数构型进行随机搜索

对于非监督式学习(聚类)：

GridSearcherSelf: 利用给定模型选择标准(11/19：轮廓系数)，对所有可能参数构型进行随机搜索

RandSearcherSelf: 利用给定模型选择标准(11/19：轮廓系数)，对所有可能参数构型进行随机搜索

是否自动寻参作为每个函数的传递参量，默认为否。

目前支持自动寻参的函数有：

0. features\_importance (11/19)

1. ML\_Classification (11/19)

2. ML\_Regression (11/19)

3. ML\_Clustering (11/19)

4. two\_groups\_classfication\_multimodels (11/19)

尚不支持自动寻参的是：

1. model\_score\_with\_features\_add

2020/12/12: 所有本文件中函数

输入参数统一增加 savePath 用来存储图片

返回形式统一为：

df\_dict, str\_result, plot\_name\_list

分别为：dataframe从名字到内容的字典，字符串输出，图片文件名列表

2020/12/31:

1. features\_importance 入参增加 bool::standardization (数据标准化, default = True)

2. model\_score\_with\_features\_add 入参增加 bool::importance\_first (先进行XGBoost重要度排序，default = True)

3. two\_groups\_classfication\_multimodels 入参删除 4 个 (ylim\_min, ylim\_max, title1, title2)，后端views.py已修改

4. 所有图片保存前增填 savePath != None 的检查; savePath = None 作为内部调用使用，不额外保存图片

2021/01/13:

1. 所有函数增加变量筛选(调用 utils\_ml.ML\_assistant 中的 filtering 函数)

2021/02/09:

1. two\_class\_multimodel 森林图调用X5 forest\_plot()

2021/05/05:

前端修改要求：

feature\_importance 的 Advanced/views.py 中， model\_type 与方法的对应编号修改如下:

1 -- 回归/Lasso

2 -- 回归/岭回归

3 -- 回归/XGBoost

4 -- 回归/随机森林

5 -- 回归/AdaBoost

6 -- 分类/Logistic

7 -- 分类/XGBoost

8 -- 分类/随机森林

9 -- 分类/AdaBoost

Classification 增加一个 dataframe

"""

import os

import random

import datetime

import pickle

import math

import pandas as pd

import numpy as np

import matplotlib

matplotlib.use('AGG')

import matplotlib.pyplot as plt

from sklearn.base import clone

from sklearn.preprocessing import StandardScaler

from sklearn.inspection import permutation\_importance

from sklearn.model\_selection import cross\_val\_predict

from sklearn.model\_selection import cross\_validate

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

from sklearn.model\_selection import KFold

from sklearn.model\_selection import StratifiedKFold

from sklearn.model\_selection import ShuffleSplit

from sklearn.model\_selection import RandomizedSearchCV

from xgboost import XGBRegressor

from sklearn.linear\_model import LassoCV, RidgeCV

from sklearn.linear\_model import LogisticRegression, LogisticRegressionCV

from sklearn.linear\_model import LinearRegression

from sklearn.ensemble import RandomForestRegressor

from sklearn.ensemble import AdaBoostRegressor

from sklearn.svm import LinearSVR

from sklearn.neighbors import KNeighborsRegressor

from sklearn.mixture import GaussianMixture

from sklearn.cluster import Birch

from sklearn.cluster import KMeans

from sklearn.cluster import AffinityPropagation

from sklearn.cluster import SpectralClustering

from sklearn.cluster import AgglomerativeClustering

from lightgbm import LGBMClassifier

from xgboost import XGBClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.ensemble import AdaBoostClassifier

from sklearn.ensemble import BaggingClassifier

from sklearn.ensemble import ExtraTreesClassifier

from sklearn.neighbors import KNeighborsClassifier

from sklearn.neural\_network import MLPClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.naive\_bayes import ComplementNB

from sklearn.svm import SVC

from sklearn.feature\_selection import RFECV

from sklearn.decomposition import PCA

from AnalysisFunction.utils\_ml.FeatrueSelect import mrmr\_classif

from AnalysisFunction.utils\_ml.FeatrueSelect import ReliefF

from sklearn.metrics import auc

from sklearn.metrics import silhouette\_score, calinski\_harabasz\_score, mutual\_info\_score, v\_measure\_score, \

normalized\_mutual\_info\_score

from sklearn.metrics import brier\_score\_loss

from sklearn.calibration import CalibratedClassifierCV, calibration\_curve

from sklearn.preprocessing import label\_binarize

from xgboost import plot\_importance

import AnalysisFunction.X\_5\_SmartPlot as x5

from AnalysisFunction.X\_5\_SmartPlot import plot\_calibration\_curve

from AnalysisFunction.X\_5\_SmartPlot import calculate\_net\_benefit

from AnalysisFunction.X\_5\_SmartPlot import plot\_decision\_curves

from AnalysisFunction.X\_1\_DataGovernance import data\_standardization

from AnalysisFunction.X\_1\_DataGovernance import \_analysis\_dict

from AnalysisFunction.X\_2\_DataSmartStatistics import comprehensive\_smart\_analysis

from AnalysisFunction.utils\_ml import filtering, dic2str, round\_dec, save\_fig

from AnalysisFunction.utils\_ml import classification\_metric\_evaluate, regression\_metric\_evaluate

from AnalysisFunction.utils\_ml import make\_class\_metrics\_dict, make\_regr\_metrics\_dict, multiclass\_metric\_evaluate

from AnalysisFunction.utils\_ml import ci

from AnalysisFunction.utils\_ml import GridSearcherCV, RandSearcherCV, GridSearcherSelf, RandSearcherSelf

from AnalysisFunction.utils\_ml.params import RandDefaultRange

from AnalysisFunction.utils\_ml.auc\_delong import delong\_roc\_test

from functools import reduce

plt.rcParams['font.sans-serif'] = ['SimHei'] # 用来正常显示中文标签

plt.rcParams['axes.unicode\_minus'] = False # 用来正常显示负号)

from matplotlib import rc

plt.rcParams['ps.useafm'] = True

rc('font', \*\*{'family': 'sans-serif', 'sans-serif': ['FreeSans']})

plt.rcParams['pdf.fonttype'] = 42

def ML\_Classfication(

df,

group,

features,

decimal\_num=3,

validation\_ratio=0.15,

scoring='roc\_auc',

method='KNeighborsClassifier',

n\_splits=10,

explain=True,

shapSet=2,

explain\_numvar=2,

explain\_sample=2,

searching=False,

validationCurve=False,

smooth=False,

savePath=None,

dpi=600,

picFormat='jpeg',

label='LABEL',

trainSet=False,

modelSave=True,

trainLabel=0,

randomState=1,

resultType=0,

\*\*kwargs,

):

"""

机器学习分类分析

Input:

df\_input:DataFrame 输入的待处理数据

group\_name:str 分组名

validation\_ratio:float 测试集比例

scoring:str 目标评价指标

method:str 使用的机器学习分类方法/模型

'LogisticRegression':LogisticRegression(\*\*kwargs),

'XGBClassifier':XGBClassifier(\*\*kwargs),

'RandomForestClassifier':RandomForestClassifier(\*\*kwargs),

'SVC':SVC(\*\*kwargs),

'KNeighborsClassifier':KNeighborsClassifier(\*\*kwargs),

n\_splits:int 交叉验证的子集数目

explain:bool 是否进行模型解释

explain\_numvar:int 需要解释的变量数

explain\_sample:int 需要例释的样本数

searching:bool 是否进行自动寻参，默认为否

savePath:str 图片存储路径

\*\*kwargs:dict 使用机器学习分类方法的参数

Return:

df\_dict: dataframe字典，包含：

df\_train\_result: 模型在训练集上的表现

df\_test\_result: 模型在测试集上的表现

str\_result: 分析结果综述

plot\_name\_list: 图片文件名列表

"""

name\_dict = {

'LogisticRegression': 'logistic',

'XGBClassifier': 'XGBoost',

'RandomForestClassifier': 'RandomForest',

'LGBMClassifier': 'LightGBM',

'SVC': 'SVM',

'MLPClassifier': 'MLP',

'GaussianNB': 'GNB',

'ComplementNB': 'CNB',

'AdaBoostClassifier': 'AdaBoost',

'KNeighborsClassifier': 'KNN',

'DecisionTreeClassifier': 'DecisionTree',

'BaggingClassifier': 'Bagging',

}

colors = x5.CB91\_Grad\_BP

str\_time = str(datetime.datetime.now().hour) + str(datetime.datetime.now().minute) + str(

datetime.datetime.now().second)

random\_number = random.randint(1, 100)

str\_time = str\_time + str(random\_number)

list\_name = [group]

plot\_name\_dict\_save = {} ##存储图片

result\_model\_save = {} ##模型存储

resThreshold = 0 ##用于存储最终的阈值

conf\_dic\_train, conf\_dic\_valid, conf\_dic\_test = {}, {}, {}

if trainSet:

df = df[features + [group] + [label]].dropna()

for fea in features:

if fea == label or label == group:

return {'error': '标签列不能在所在模型中，请重新选择数据划分标签列！'}

else:

df = df[features + [group]].dropna()

binary = True

u = np.sort(np.unique(np.array(df[group])))

if len(u) == 2 and set(u) != set([0, 1]):

y\_result = label\_binarize(df[group], classes=[ii for ii in u]) # 将标签二值化

y\_result\_pd = pd.DataFrame(y\_result, columns=[group])

df = pd.concat([df.drop(group, axis=1), y\_result\_pd], axis=1)

elif len(u) > 2:

if len(u) > 10:

return {'error': '暂不允许类别数目大于10的情况。请检查因变量取值情况。'}

binary = False

if scoring == 'roc\_auc':

scoring = scoring + '\_ovo'

else:

scoring = scoring + '\_macro'

return {'error': '暂时只支持二分类。请检查因变量取值情况。'}

if trainSet:

if isinstance(df[label][0], str):

trainLabel = str(trainLabel)

df = df[features + [group] + [label]].dropna()

train\_a = df[df[label] == trainLabel]

test\_a = df[df[label] != trainLabel]

train\_all = train\_a.drop(label, axis=1)

test\_all = test\_a.drop(label, axis=1)

# features.remove(fea)

df = df.drop(label, axis=1)

Xtrain = train\_all.drop(group, axis=1)

Ytrain = train\_all.loc[:, list\_name].squeeze(axis=1)

Xtest = test\_all.drop(group, axis=1)

Ytest = test\_all.loc[:, list\_name].squeeze(axis=1)

else:

df = df[features + [group]].dropna()

X = df.drop(group, axis=1)

Y = df.loc[:, list\_name].squeeze(axis=1)

Xtrain, Xtest, Ytrain, Ytest = TTS(X, Y, test\_size=validation\_ratio, random\_state=randomState, )

df\_dict = {}

str\_result = "采用%s机器学习方法进行分类，分类变量为%s，模型中的变量包括" % (method, group)

str\_result += '、'.join(features)

if searching == True:

if method == 'LGBMClassifier':

searcher = GridSearcherCV('Classification', globals()[method]())

clf = searcher(Xtrain, Ytrain);

searcher.report()

else:

searcher = RandSearcherCV('Classification', globals()[method]())

clf = searcher(Xtrain, Ytrain);

searcher.report()

elif searching == 'Handle':

if (method == 'SVC'): kwargs['probability'] = True

if (method == 'RandomForestClassifier' and kwargs['max\_depth'] == 'None'):

kwargs['max\_depth'] = None

if (method == 'MLPClassifier'):

hls\_vals = str(kwargs['hidden\_layer\_sizes']).split(',')

hls\_value = ()

for hls\_val in hls\_vals:

try:

if int(hls\_val) >= 5 and int(hls\_val) <= 200:

hls\_value = hls\_value + (int(hls\_val),)

else:

return {'error': '请按照要求重新设置隐藏层宽度！'}

except:

return {'error': '请重新设神经网络模型中的隐藏层宽度！'}

kwargs['hidden\_layer\_sizes'] = hls\_value

if (method == 'GaussianNB' and kwargs['priors'] == 'None'):

kwargs['priors'] = None

elif (method == 'GaussianNB'):

pri\_vals = str(kwargs['priors']).split(',')

pri\_value = ()

pri\_sum = 0.0

for pri\_val in pri\_vals:

try:

pri\_sum = float(pri\_val) + pri\_sum

pri\_value = pri\_value + (float(pri\_val),)

except:

return {'error': '请重新设朴素贝叶斯模型中的先验概率！'}

if len(pri\_vals) == len(Y.unique()) and pri\_sum == 1.0:

kwargs['priors'] = pri\_value

else:

return {'error': '请重新设朴素贝叶斯模型中的先验概率！'}

clf = globals()[method](\*\*kwargs).fit(Xtrain, Ytrain)

elif searching == False:

# if (method == 'SVC'): kwargs['probability'] = True

if method == 'SVC':

kwargs['probability'] = True

elif method == 'MLPClassifier':

kwargs['hidden\_layer\_sizes'] = (20, 10)

kwargs['max\_iter'] = 20

elif method == 'RandomForestClassifier':

kwargs['n\_estimators'] = 20

clf = globals()[method](\*\*kwargs).fit(Xtrain, Ytrain)

str\_result += "\n模型参数为:\n%s" % dic2str(clf.get\_params(), clf.\_\_class\_\_.\_\_name\_\_)

str\_result += "\n数据集样本数总计N=%d例，分类变量中包含的类别信息为：\n" % (df.shape[0])

group\_labels = df[group].unique()

group\_labels.sort()

for label in group\_labels:

n = sum(df[group] == label)

str\_result += "\t 类别(" + str(label) + ")：N=" + str(n) + "例\n"

plot\_name\_list = x5.plot\_learning\_curve(

clf,

Xtrain,

Ytrain,

cv=n\_splits,

scoring=scoring,

path=savePath,

dpi=dpi,

picFormat=picFormat,

)

plot\_name\_dict\_save['学习曲线'] = plot\_name\_list[1]

plot\_name\_list.pop(len(plot\_name\_list) - 1)

###画校准曲线

calibration\_curve\_name, \_ = plot\_calibration\_curve(clf, Xtrain, Xtest, Ytrain, Ytest, name=method, path=savePath,

smooth=smooth,

picFormat=picFormat, dpi=dpi, )

plot\_name\_list.append(calibration\_curve\_name[0])

plot\_name\_dict\_save['校准曲线'] = calibration\_curve\_name[1]

if binary:

fig = plt.figure(figsize=(4, 4), dpi=dpi)

# 画对角线

plt.plot(

[0, 1], [0, 1],

linestyle='--',

lw=1, color='r',

alpha=0.8,

)

plt.grid(which='major', axis='both', linestyle='-.', alpha=0.08, color='grey')

best\_auc = 0.0

tprs\_train, tprs\_valid = [], []

fpr\_train\_alls, tpr\_train\_alls = [], []

mean\_fpr = np.linspace(0, 1, 100)

list\_evaluate\_dic\_train = make\_class\_metrics\_dict()

list\_evaluate\_dic\_valid = make\_class\_metrics\_dict()

# KF = KFold(n\_splits=n\_splits, random\_state=randomState,shuffle=True)##StratifiedKFold

KF = StratifiedKFold(n\_splits=n\_splits, random\_state=randomState, shuffle=True)

for i, (train\_index, valid\_index) in enumerate(KF.split(Xtrain, Ytrain)):

# 划分训练集和验证集

X\_train, X\_valid = Xtrain.iloc[train\_index], Xtrain.iloc[valid\_index]

Y\_train, Y\_valid = Ytrain.iloc[train\_index], Ytrain.iloc[valid\_index]

# 建立模型(模型已经定义)并训练

model = clone(clf).fit(X\_train, Y\_train)

# 利用classification\_metric\_evaluate函数获取在验证集的预测值

fpr\_train, tpr\_train, metric\_dic\_train, \_ = classification\_metric\_evaluate(model, X\_train, Y\_train, binary)

fpr\_valid, tpr\_valid, metric\_dic\_valid, \_ = classification\_metric\_evaluate(model, X\_valid, Y\_valid, binary,

Threshold=metric\_dic\_train['cutoff'])

metric\_dic\_valid.update({'cutoff': metric\_dic\_train['cutoff']})

# model selection using validation set

if metric\_dic\_valid['AUC'] > best\_auc:

clf = model

resThreshold = metric\_dic\_train['cutoff']

# 计算所有评价指标

for key in list\_evaluate\_dic\_train.keys():

list\_evaluate\_dic\_train[key].append(metric\_dic\_train[key])

list\_evaluate\_dic\_valid[key].append(metric\_dic\_valid[key])

if binary:

# interp:插值 把结果添加到tprs列表中

tprs\_valid.append(np.interp(mean\_fpr, fpr\_valid, tpr\_valid))

tprs\_valid[-1][0] = 0.0

# 画图, 只需要plt.plot(fpr,tpr), 变量roc\_auc只是记录auc的值, 通过auc()函数计算出来

if validationCurve:

plt.plot(

fpr\_valid, tpr\_valid,

lw=1, alpha=0.4,

label='ROC fold %4d (auc=%0.3f 95%%CI (%0.3f-%0.3f))' % (

i + 1, metric\_dic\_valid['AUC'], metric\_dic\_valid['AUC\_L'], metric\_dic\_valid['AUC\_U']),

)

##训练集ROC

fpr\_train\_alls.append(fpr\_train)

tpr\_train\_alls.append(tpr\_train)

tprs\_train.append(np.interp(mean\_fpr, fpr\_train, tpr\_train))

tprs\_train[-1][0] = 0.0

if modelSave:

import pickle

modelfile = open(savePath + method + str\_time + '.pkl', 'wb')

pickle.dump(clf, modelfile)

modelfile.close()

result\_model\_save['modelFile'] = method + str\_time + '.pkl'

result\_model\_save['modelFeature'] = features

if binary:

mean\_tpr\_valid = np.mean(tprs\_valid, axis=0)

mean\_tpr\_valid[-1] = 1.0

mean\_auc = auc(mean\_fpr, mean\_tpr\_valid) # 计算平均AUC值

aucs\_lower, aucs\_upper = ci(list\_evaluate\_dic\_valid['AUC'])

plt.plot(

mean\_fpr, mean\_tpr\_valid,

color='b',

lw=2, alpha=0.8,

label=r'Mean (validation) ROC (auc=%0.3f 95%%CI (%0.3f-%0.3f))' % (

mean\_auc, np.mean(list\_evaluate\_dic\_valid['AUC\_L']), np.mean(list\_evaluate\_dic\_valid['AUC\_U'])),

# label = r'Mean ROC (auc=%0.3f 0.95CI(%0.3f-%0.3f)' % (mean\_auc, aucs\_lower, aucs\_upper),

)

mean\_dic\_train, stdv\_dic\_train = {}, {}

mean\_dic\_valid, stdv\_dic\_valid = {}, {}

for key in list\_evaluate\_dic\_valid.keys():

mean\_dic\_train[key] = np.mean(list\_evaluate\_dic\_train[key])

mean\_dic\_valid[key] = np.mean(list\_evaluate\_dic\_valid[key])

if resultType == 0: ##SD

stdv\_dic\_train[key] = np.std(list\_evaluate\_dic\_train[key], axis=0)

stdv\_dic\_valid[key] = np.std(list\_evaluate\_dic\_valid[key], axis=0)

elif resultType == 1: ##CI

conf\_dic\_train[key] = list(ci(list\_evaluate\_dic\_train[key]))

conf\_dic\_valid[key] = list(ci(list\_evaluate\_dic\_valid[key]))

# if resultType == 0: ##SD

# df\_train\_result = pd.DataFrame([mean\_dic\_train, stdv\_dic\_train], index=['Mean', 'SD'])

# df\_train\_result = df\_train\_result.applymap(lambda x: round\_dec(x, d=decimal\_num))

# df\_valid\_result = pd.DataFrame([mean\_dic\_valid, stdv\_dic\_valid], index=['Mean', 'SD'])

# df\_valid\_result = df\_valid\_result.applymap(lambda x: round\_dec(x, d=decimal\_num))

fpr\_test, tpr\_test, metric\_dic\_test, df\_test\_result = classification\_metric\_evaluate(clf, Xtest, Ytest, binary,

Threshold=resThreshold)

metric\_dic\_test.update({'cutoff': resThreshold})

# plt.plot(

# fpr\_test, tpr\_test,

# lw=1.5, alpha=0.6,

# label='Test Set ROC (auc=%0.3f) ' % metric\_dic\_test['AUC'],

# )

plt.xlim([-0.02, 1.02])

plt.ylim([-0.02, 1.02])

plt.xlabel('1-Specificity')

plt.ylabel('Sensitivity')

plt.title('Validation ROC')

plt.legend(loc='lower right', fontsize=5)

if savePath is not None:

plot\_name\_list.append(save\_fig(savePath, 'ROC\_curve', 'png', fig, str\_time=str\_time))

plot\_name\_dict\_save['验证集ROC曲线'] = save\_fig(savePath, 'ROC\_curve', picFormat, fig, str\_time=str\_time)

plt.close()

##画训练集ROC

if binary:

fig = plt.figure(figsize=(4, 4), dpi=dpi)

# 画对角线

plt.plot(

[0, 1], [0, 1],

linestyle='--',

lw=1, color='r',

alpha=0.8,

)

plt.grid(which='major', axis='both', linestyle='-.', alpha=0.08, color='grey')

if validationCurve:

for i in range(len(tpr\_train\_alls)):

plt.plot(

fpr\_train\_alls[i], tpr\_train\_alls[i],

lw=1, alpha=0.4,

label='ROC fold %4d (auc=%0.3f 95%%CI (%0.3f-%0.3f)) ' % (

i + 1, list\_evaluate\_dic\_train['AUC'][i], list\_evaluate\_dic\_train['AUC\_L'][i],

list\_evaluate\_dic\_train['AUC\_U'][i]),

)

mean\_tpr\_train = np.mean(tprs\_train, axis=0)

mean\_tpr\_train[-1] = 1.0

mean\_auc\_train = auc(mean\_fpr, mean\_tpr\_train) # 计算平均AUC值

plt.plot(

mean\_fpr, mean\_tpr\_train,

color='b',

lw=1.8, alpha=0.7,

label=r'Mean (train) ROC (auc=%0.3f 95%%CI (%0.3f-%0.3f))' % (

mean\_auc\_train, np.mean(list\_evaluate\_dic\_train['AUC\_L']), np.mean(list\_evaluate\_dic\_train['AUC\_U'])),

# label = r'Mean ROC (auc=%0.3f 0.95CI(%0.3f-%0.3f)' % (mean\_auc, aucs\_lower, aucs\_upper),

)

plt.xlim([-0.02, 1.02])

plt.ylim([-0.02, 1.02])

plt.xlabel('1-Specificity')

plt.ylabel('Sensitivity')

plt.title('Train ROC')

plt.legend(loc='lower right', fontsize=5)

if savePath is not None:

plot\_name\_list.append(save\_fig(savePath, 'ROC\_curve\_train', 'png', fig, str\_time=str\_time))

plot\_name\_dict\_save['训练集ROC曲线'] = save\_fig(savePath, 'ROC\_curve\_train', picFormat, fig, str\_time=str\_time)

plt.close()

plot\_name\_list.reverse() ###所有图片倒置

###画测试集ROC

fig = plt.figure(figsize=(4, 4), dpi=dpi)

# 画对角线

plt.plot(

[0, 1], [0, 1],

linestyle='--',

lw=1, color='r',

alpha=0.8,

)

plt.grid(which='major', axis='both', linestyle='-.', alpha=0.08, color='grey')

if smooth:

from scipy.interpolate import interp1d

tpr\_test\_unique, tpr\_test\_index = np.unique(fpr\_test, return\_index=True)

fpr\_test\_new = np.linspace(min(fpr\_test), max(fpr\_test), len(fpr\_test))

f = interp1d(tpr\_test\_unique, tpr\_test[tpr\_test\_index], kind='linear') ##cubic

tpr\_test\_new = f(fpr\_test\_new)

else:

fpr\_test\_new = fpr\_test

tpr\_test\_new = tpr\_test

plt.plot(

fpr\_test\_new, tpr\_test\_new,

lw=1.5, alpha=0.6, color='b',

label='Test Set ROC (auc=%0.3f 95%%CI (%0.3f-%0.3f)) ' % (

metric\_dic\_test['AUC'], metric\_dic\_test['AUC\_L'], metric\_dic\_test['AUC\_U']),

)

plt.xlim([-0.02, 1.02])

plt.ylim([-0.02, 1.02])

plt.xlabel('1-Specificity')

plt.ylabel('Sensitivity')

plt.title('Test ROC')

plt.legend(loc='lower right', fontsize=5)

if savePath is not None:

plot\_name\_list.append(save\_fig(savePath, 'ROC\_curve\_test', 'png', fig, str\_time=str\_time))

plot\_name\_dict\_save['测试集ROC曲线'] = save\_fig(savePath, 'ROC\_curve\_test', picFormat, fig, str\_time=str\_time)

plt.close()

# df\_test\_result = df\_test\_result.applymap(lambda x: round\_dec(x, d=decimal\_num))

if trainSet:

df\_count\_c = Xtest.shape[0]

df\_count\_r = (Xtest.shape[0] / df.shape[0]) \* 100

str\_result += '其中在总体样本中根据标签%s为%s划分为总体训练集，其余标签归为测试集。其中' % (label, trainLabel)

else:

df\_count\_c = df.shape[0] \* validation\_ratio

df\_count\_r = validation\_ratio \* 100

str\_result += '其中在总体样本中随机抽取'

diff, ratio = 0, 0

if resultType == 1: ##CI

str\_result += "测试集N=%d例(%3.2f%%)，剩余样本作为训练集进行%d折交叉验证，并在验证集中得到AUC=%5.4f(%5.4f-%5.4f)。\n最终模型在测试集中的AUC=%5.4f，准确度=%5.4f。\n" % (

df\_count\_c,

df\_count\_r,

n\_splits,

mean\_dic\_valid['AUC'],

mean\_dic\_valid['AUC\_L'],

mean\_dic\_valid['AUC\_U'],

df\_test\_result['AUC'].values[0],

df\_test\_result['准确度'].values[0]

)

diff = mean\_dic\_valid['AUC'] - float(df\_test\_result.loc['Mean', 'AUC'])

ratio = diff / float(df\_test\_result.loc['Mean', 'AUC'])

elif resultType == 0: ##SD

str\_result += "取测试集N=%d例(%3.2f%%)，剩余样本作为训练集进行%d折交叉验证，并在验证集中得到AUC=%5.4f±%5.4f。\n最终模型在测试集中的AUC=%5.4f，准确度=%5.4f。\n" % (

df\_count\_c,

df\_count\_r,

n\_splits,

mean\_dic\_valid['AUC'],

stdv\_dic\_valid['AUC'],

df\_test\_result['AUC'].values[0],

df\_test\_result['准确度'].values[0]

)

diff = float(stdv\_dic\_valid['AUC']) - float(df\_test\_result.loc['Mean', 'AUC'])

ratio = diff / float(df\_test\_result.loc['Mean', 'AUC'])

if (not np.isnan(float(diff)) and diff > 0 and (ratio > 0.1)):

str\_result += '注意到AUC指标下验证集表现超出测试集{}，约{}%，可能存在过拟合现象。建议更换模型或重新设置参数。'.format(round(diff, decimal\_num),

round(ratio \* 100, decimal\_num))

else:

str\_result += '鉴于AUC指标下验证集表现未超出测试集或超出比小于10%，可认为拟合成功，{}模型可以用于此数据集的分类建模任务。'.format(name\_dict[method])

str\_result += '\n如果想进一步对比更多分类模型的表现，可使用左侧栏智能分析中的‘分类多模型综合分析’功能。'

df\_test\_result = df\_test\_result.applymap(lambda x: round\_dec(x, d=decimal\_num))

if resultType == 1: ##CI

for tem in ['AUC', 'AUC\_L', 'AUC\_U']:

del conf\_dic\_train[tem]

del conf\_dic\_valid[tem]

for key in conf\_dic\_train.keys():

mean\_dic\_train[key] = str(round\_dec(float(mean\_dic\_train[key]), d=decimal\_num)) + '(' + \

str(round\_dec(float(conf\_dic\_train[key][0]), d=decimal\_num)) + '-' + \

str(round\_dec(float(conf\_dic\_train[key][1]), d=decimal\_num)) + ')'

mean\_dic\_valid[key] = str(round\_dec(float(mean\_dic\_valid[key]), d=decimal\_num)) + '(' + \

str(round\_dec(float(conf\_dic\_valid[key][0]), d=decimal\_num)) + '-' + \

str(round\_dec(float(conf\_dic\_valid[key][1]), d=decimal\_num)) + ')'

df\_train\_result = pd.DataFrame([mean\_dic\_train], index=['Mean'])

# df\_train\_result = df\_train\_result.applymap(lambda x: round\_dec(x, d=decimal\_num))

df\_valid\_result = pd.DataFrame([mean\_dic\_valid], index=['Mean'])

df\_train\_result.iloc[0, 0] = str(round\_dec(float(df\_train\_result.iloc[0, 0]), d=decimal\_num)) + '(' + \

str(round\_dec(float(df\_train\_result.iloc[0, -2]), d=decimal\_num)) + '-' + \

str(round\_dec(float(df\_train\_result.iloc[0, -1]), d=decimal\_num)) + ')'

df\_valid\_result.iloc[0, 0] = str(round\_dec(float(df\_valid\_result.iloc[0, 0]), d=decimal\_num)) + '(' + \

str(round\_dec(float(df\_valid\_result.iloc[0, -2]), d=decimal\_num)) + '-' + \

str(round\_dec(float(df\_valid\_result.iloc[0, -1]), d=decimal\_num)) + ')'

df\_train\_result.rename(columns={"AUC": 'AUC(95%CI)', 'cutoff': 'cutoff(95%CI)', '准确度': '准确度(95%CI)',

'灵敏度': '灵敏度(95%CI)', '特异度': '特异度(95%CI)',

'阳性预测值': '阳性预测值(95%CI)', '阴性预测值': '阴性预测值(95%CI)',

'F1分数': 'F1分数(95%CI)', 'Kappa': 'Kappa(95%CI)'}, inplace=True)

df\_valid\_result.rename(columns={"AUC": 'AUC(95%CI)', 'cutoff': 'cutoff(95%CI)', '准确度': '准确度(95%CI)',

'灵敏度': '灵敏度(95%CI)', '特异度': '特异度(95%CI)',

'阳性预测值': '阳性预测值(95%CI)', '阴性预测值': '阴性预测值(95%CI)',

'F1分数': 'F1分数(95%CI)', 'Kappa': 'Kappa(95%CI)'}, inplace=True)

df\_test\_result.iloc[0, 0] = str(df\_test\_result.iloc[0, 0]) + ' (' + str(df\_test\_result.iloc[0, -2]) + '-' + str(

df\_test\_result.iloc[0, -1]) + ')'

df\_test\_result.rename(columns={"AUC": 'AUC (95%CI)'}, inplace=True)

elif resultType == 0: ##SD

for tem in ['AUC\_L', 'AUC\_U']:

del stdv\_dic\_train[tem]

del stdv\_dic\_valid[tem]

for key in stdv\_dic\_train.keys():

mean\_dic\_train[key] = str(round\_dec(float(mean\_dic\_train[key]), d=decimal\_num)) + ' (' + \

str(round\_dec(float(stdv\_dic\_train[key]), d=decimal\_num)) + ')'

mean\_dic\_valid[key] = str(round\_dec(float(mean\_dic\_valid[key]), d=decimal\_num)) + ' (' + \

str(round\_dec(float(stdv\_dic\_valid[key]), d=decimal\_num)) + ')'

df\_train\_result = pd.DataFrame([mean\_dic\_train], index=['Mean'])

df\_valid\_result = pd.DataFrame([mean\_dic\_valid], index=['Mean'])

df\_train\_result.rename(columns={"AUC": 'AUC(SD)', 'cutoff': 'cutoff(SD)', '准确度': '准确度(SD)',

'灵敏度': '灵敏度(SD)', '特异度': '特异度(SD)',

'阳性预测值': '阳性预测值(SD)', '阴性预测值': '阴性预测值(SD)',

'F1分数': 'F1分数(SD)', 'Kappa': 'Kappa(SD)'}, inplace=True)

df\_valid\_result.rename(columns={"AUC": 'AUC(SD)', 'cutoff': 'cutoff(SD)', '准确度': '准确度(SD)',

'灵敏度': '灵敏度(SD)', '特异度': '特异度(SD)',

'阳性预测值': '阳性预测值(SD)', '阴性预测值': '阴性预测值(SD)',

'F1分数': 'F1分数(SD)', 'Kappa': 'Kappa(SD)'}, inplace=True)

df\_dictjq = {

'训练集结果汇总': df\_train\_result.iloc[0:2, 0:8],

'验证集结果汇总': df\_valid\_result.iloc[0:2, 0:8],

'测试集结果汇总': df\_test\_result.iloc[0:2, 0:8],

}

df\_dict.update(df\_dictjq)

plot\_name\_dict = {

'训练集ROC曲线图': plot\_name\_list[0],

'验证集ROC曲线图': plot\_name\_list[1],

'测试集ROC曲线图': plot\_name\_list[4],

'学习曲线图': plot\_name\_list[3],

'模型校准曲线': plot\_name\_list[2],

}

if binary: ###画DCA曲线

DCA\_dict = {}

prob\_pos, p\_serie, net\_benefit\_serie, net\_benefit\_serie\_All = calculate\_net\_benefit(clf, Xtest, Ytest)

DCA\_dict[name\_dict[method]] = {'p\_serie': p\_serie, 'net\_b\_s': net\_benefit\_serie,

'net\_b\_s\_A': net\_benefit\_serie\_All}

decision\_curve\_p = plot\_decision\_curves(DCA\_dict, colors=colors, name='Test', savePath=savePath, dpi=dpi,

picFormat=picFormat)

plot\_name\_dict['测试集DCA曲线图'] = decision\_curve\_p[0]

plot\_name\_dict\_save['测试集DCA曲线图'] = decision\_curve\_p[1]

if explain or modelSave:

import shap

# from interpret.blackbox import LimeTabular, PartialDependence

f = lambda x: clf.predict\_proba(x)[:, 1]

med = Xtrain.median().values.reshape((1, Xtrain.shape[1]))

result\_model\_save['modelShapValue'] = [float('{:.3f}'.format(i)) for i in list(med[0])] ##list(med[0])

result\_model\_save['modelName'] = method

result\_model\_save['modelClass'] = '机器学习分类'

result\_model\_save['Threshold'] = resThreshold

df\_shapValue = Xtest

df\_shapValue\_show = pd.DataFrame()

shapValue\_list = []

shapValue\_name = []

if explain:

if shapSet == 2: ##Xtrain, Xtest, Ytrain, Ytest

df\_shapValue = Xtest

if (explain\_sample == 4):

flage1, flage2, flage3, flage4 = True, True, True, True

for i in range(len(Ytest)):

if (flage1 and f(df\_shapValue.iloc[i:i + 1, :])[0] >= resThreshold and Ytest.iloc[i,] == 1):

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[i:i + 1, :]], axis=0)

shapValue\_list.append(i)

shapValue\_name.append('shap\_样本\_预测值为1实际值为1')

flage1 = False

elif (flage2 and f(df\_shapValue.iloc[i:i + 1, :])[0] >= resThreshold and Ytest.iloc[i,] == 0):

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[i:i + 1, :]], axis=0)

shapValue\_list.append(i)

shapValue\_name.append('shap\_样本\_预测值为1实际值为0')

flage2 = False

elif (flage3 and f(df\_shapValue.iloc[i:i + 1, :])[0] < resThreshold and Ytest.iloc[i,] == 1):

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[i:i + 1, :]], axis=0)

shapValue\_name.append('shap\_样本\_预测值为0实际值为1')

shapValue\_list.append(i)

flage3 = False

elif (flage4 and f(df\_shapValue.iloc[i:i + 1, :])[0] < resThreshold and Ytest.iloc[i,] == 0):

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[i:i + 1, :]], axis=0)

shapValue\_name.append('shap\_样本\_预测值为0实际值为0')

shapValue\_list.append(i)

flage4 = False

if (not flage1 and not flage2 and not flage3 and not flage4):

break

else:

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[0:explain\_sample, :]], axis=0)

shapValue\_list.extend(i for i in range(explain\_sample))

shapValue\_name.extend('shap\_样本\_' + str(i) for i in range(explain\_sample))

elif shapSet == 1:

df\_shapValue = Xtrain

if (explain\_sample == 4):

flage1, flage2, flage3, flage4 = True, True, True, True

for i in range(len(Ytrain)):

if (flage1 and f(df\_shapValue.iloc[i:i + 1, :])[0] >= resThreshold and Ytrain.iloc[i,] == 1):

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[i:i + 1, :]], axis=0)

shapValue\_name.append('shap\_样本\_预测值为1实际值为1')

shapValue\_list.append(i)

flage1 = False

elif (flage2 and f(df\_shapValue.iloc[i:i + 1, :])[0] >= resThreshold and Ytrain.iloc[i,] == 0):

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[i:i + 1, :]], axis=0)

shapValue\_name.append('shap\_样本\_预测值为1实际值为0')

shapValue\_list.append(i)

flage2 = False

elif (flage3 and f(df\_shapValue.iloc[i:i + 1, :])[0] < resThreshold and Ytrain.iloc[i,] == 1):

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[i:i + 1, :]], axis=0)

shapValue\_name.append('shap\_样本\_预测值为0实际值为1')

shapValue\_list.append(i)

flage3 = False

elif (flage4 and f(df\_shapValue.iloc[i:i + 1, :])[0] < resThreshold and Ytrain.iloc[i,] == 0):

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[i:i + 1, :]], axis=0)

shapValue\_name.append('shap\_样本\_预测值为0实际值为0')

shapValue\_list.append(i)

flage4 = False

if (not flage1 and not flage2 and not flage3 and not flage4):

break

else:

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[0:explain\_sample, :]], axis=0)

shapValue\_list.extend(i for i in range(explain\_sample))

shapValue\_name.extend('shap\_样本\_' + str(i) for i in range(explain\_sample))

elif shapSet == 0:

df\_shapValue = pd.concat([Xtrain, Xtest], axis=0)

df\_shapValue\_Y = pd.concat([Ytrain, Ytest], axis=0)

if (explain\_sample == 4):

flage1, flage2, flage3, flage4 = True, True, True, True

for i in range(len(df\_shapValue\_Y)):

if (flage1 and f(df\_shapValue.iloc[i:i + 1, :])[0] >= resThreshold and df\_shapValue\_Y.iloc[

i,] == 1):

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[i:i + 1, :]], axis=0)

shapValue\_name.append('shap\_样本\_预测值为1实际值为1')

shapValue\_list.append(i)

flage1 = False

elif (flage2 and f(df\_shapValue.iloc[i:i + 1, :])[0] >= resThreshold and df\_shapValue\_Y.iloc[

i,] == 0):

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[i:i + 1, :]], axis=0)

shapValue\_name.append('shap\_样本\_预测值为1实际值为0')

shapValue\_list.append(i)

flage2 = False

elif (flage3 and f(df\_shapValue.iloc[i:i + 1, :])[0] < resThreshold and df\_shapValue\_Y.iloc[

i,] == 1):

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[i:i + 1, :]], axis=0)

shapValue\_name.append('shap\_样本\_预测值为0实际值为1')

shapValue\_list.append(i)

flage3 = False

elif (flage4 and f(df\_shapValue.iloc[i:i + 1, :])[0] < resThreshold and df\_shapValue\_Y.iloc[

i,] == 0):

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[i:i + 1, :]], axis=0)

shapValue\_name.append('shap\_样本\_预测值为0实际值为0')

shapValue\_list.append(i)

flage4 = False

if (not flage1 and not flage2 and not flage3 and not flage4):

break

else:

df\_shapValue\_show = pd.concat([df\_shapValue\_show, df\_shapValue.iloc[0:explain\_sample, :]], axis=0)

shapValue\_list.extend(i for i in range(explain\_sample))

shapValue\_name.extend('shap\_样本\_' + str(i) for i in range(explain\_sample))

explainer = shap.KernelExplainer(f, med)

shap\_values = explainer.shap\_values(df\_shapValue)

if explain\_numvar > 0:

# SHAP beeswarm summary plot

assert explain\_numvar <= len(features)

fig = shap.summary\_plot(shap\_values, df\_shapValue, show=False)

if savePath is not None:

plot\_name\_dict['SHAP\_变量贡献度总结'] = save\_fig(savePath, 'shap\_summary', 'png', fig, str\_time=str\_time)

plot\_name\_dict\_save['SHAP\_变量贡献度总结'] = save\_fig(savePath, 'shap\_summary', picFormat, fig,

str\_time=str\_time)

plt.close(fig)

fig1 = shap.summary\_plot(shap\_values, df\_shapValue, plot\_type='bar', show=False)

if savePath is not None:

plot\_name\_dict['SHAP\_重要性图'] = save\_fig(savePath, 'shap\_import', 'png', fig1, str\_time=str\_time)

plot\_name\_dict\_save['SHAP\_重要性图'] = save\_fig(savePath, 'shap\_import', picFormat, fig1,

str\_time=str\_time)

plt.close(fig1)

# # single feature (Partial Dependence)

# pdp = PartialDependence(predict\_fn=clf.predict\_proba, data=Xtrain)

# pdp\_global = pdp.explain\_global(name='Partial Dependence')

# for i in range(explain\_numvar):

# fig = pdp\_global.visualize(key=i)

# if savePath is not None:

# plot\_name\_dict['微分依赖度\_变量{}'.format(i+1)] = save\_fig(savePath, 'partial\_dependence\_{}'.format(features[i]), '.jpeg', fig)

# plt.close()

if explain\_sample > 0:

assert explain\_sample <= len(Ytest)

# lime = LimeTabular(predict\_fn=clf.predict\_proba, data=Xtest, random\_state=1)

# lime\_local = lime.explain\_local(Xtest[:explain\_sample], Ytest[:explain\_sample], name='LIME')

for i in range(len(shapValue\_list)):

# SHAP explain

fig = shap.force\_plot(explainer.expected\_value, shap\_values[shapValue\_list[i]],

df\_shapValue\_show.iloc[i, :], show=False,

figsize=(15, 3), matplotlib=True)

if savePath is not None:

plot\_name\_dict[shapValue\_name[i]] = save\_fig(savePath, 'shap\_sample\_{}'.format(i + 1),

'png', fig, str\_time=str\_time)

plot\_name\_dict\_save[shapValue\_name[i]] = save\_fig(savePath, 'shap\_sample\_{}'.format(i + 1),

picFormat, fig, str\_time=str\_time)

plt.close(fig)

# # LIME explain

# fig = lime\_local.visualize(key=i)

# if savePath is not None:

# plot\_name\_dict['LIME\_样本{}'.format(i+1)] = save\_fig(savePath, 'lime\_{}'.format(i), '.jpeg', fig)

# plt.close()

result\_dict = {'str\_result': {'分析结果描述': str\_result}, 'tables': df\_dict,

'pics': plot\_name\_dict, 'save\_pics': plot\_name\_dict\_save,

'model': result\_model\_save}

return result\_dict

###XGBoost 重要度

def \_xgboost\_cfeatures\_importance(

df\_input,

x\_columns,

y\_column,

top\_features,

searching=True,

savePath=None,

dpi=600,

picFormat='jpeg',

):

"""

df\_input:Dataframe

x\_columns:自变量list

y\_column：因变量str

top\_features:图表中展示的特征数量

model: XGBOOST模型，如果不传则自动产生一个自动寻参后的XGBOOST模型

searching: 是否自动寻参，默认为是

savePath:str 图片存储路径

hyperparams: XGBClassifier params -- no selection yet

"""

x = df\_input[x\_columns]

y = df\_input[y\_column]

if searching:

searcher = RandSearcherCV('Classification', XGBClassifier())

model = searcher(x, y) # ; searcher.report()

else:

model = XGBClassifier()

str\_result = '采用极端梯度提升树(XGBOOST)进行变量重要度分析，模型参数为:\n' + dic2str(model.get\_params(), model.\_\_class\_\_.\_\_name\_\_)

model.fit(x, y)

col\_refs = {

'Variable': 'total\_gain',

'Total Gain': 'total\_gain',

'Total Cover': 'total\_cover',

'Gain': 'gain',

'Cover': 'cover',

'Weight Importance': 'weight',

}

df = pd.DataFrame(columns=list(col\_refs.keys()))

for col\_name, importance\_type in col\_refs.items():

row\_index = 0

for d, x in model.get\_booster().get\_score(importance\_type=importance\_type).items():

df.loc[row\_index, col\_name] = x if (col\_name != 'Variable') else d

row\_index += 1

df = df.sort\_values(by='Total Gain', ascending=False).head(top\_features)

top\_list = list(df["Variable"])

str\_result += '\n重要度最高的{}个变量（由高到低）分别为：{}。'.format(top\_features, str(top\_list)[1:-1])

plot\_name\_dict = {}

if savePath is not None:

plot\_name\_dict = x5.horizontal\_bar\_plot(

df.head(top\_features).sort\_values(by="Total Gain", ascending=True),

'Variable',

'Total Gain',

'Feature Importance (Total Gain)',

savePath,

dpi=dpi,

picFormat=picFormat,

)

return df, str\_result, plot\_name\_dict['pics'], plot\_name\_dict['save\_pics']

###随机森林重要度

def \_randomforest\_cfeatures\_importance(

df\_input,

x\_columns,

y\_column,

top\_features,

searching=True,

savePath=None,

dpi=600,

picFormat='jpeg',

):

"""

df\_input:Dataframe

x\_columns:自变量list

y\_column：因变量str

num\_features:图表中展示的特征数量

(会剔除空值)

savePath:str 图片存储路径

hyperparams: alpha(alpharange), cv, tol

"""

dftemp = df\_input[x\_columns + [y\_column]].dropna()

x = dftemp[x\_columns]

y = dftemp[y\_column]

if searching:

searcher = RandSearcherCV('Classification', RandomForestClassifier())

model = searcher(x, y) # ; searcher.report()

else:

model = RandomForestClassifier().fit(x, y)

param\_dict = model.get\_params()

str\_result = '采用Random Forrest Classifier进行变量重要度分析，模型参数为:\n' + dic2str(param\_dict, model.\_\_class\_\_.\_\_name\_\_)

df\_result = pd.DataFrame({

'Variable': x\_columns,

'Weight Importance': abs(model.feature\_importances\_)

}).sort\_values(by="Weight Importance", ascending=False)

top\_list = list(df\_result.head(top\_features)["Variable"])

str\_result += '\n重要度最高的{0}个变量（由高到低）分别为：{1}。\n'.format(top\_features, str(top\_list)[1:-1])

plot\_name\_dict = {}

if savePath is not None:

plot\_name\_dict = x5.horizontal\_bar\_plot(

df\_result.head(top\_features).sort\_values(by="Weight Importance", ascending=True),

'Variable',

'Weight Importance',

'Feature Importance (Coefficient)',

savePath,

dpi=dpi,

picFormat=picFormat,

)

return df\_result, str\_result, plot\_name\_dict['pics'], plot\_name\_dict['save\_pics']

##SVM

def \_svm\_cfeatures\_importance(

df\_input,

x\_columns,

y\_column,

top\_features,

searching=True,

savePath=None,

dpi=600,

picFormat='jpeg',

):

"""

df\_input:Dataframe

x\_columns:自变量list

y\_column：因变量str

num\_features:图表中展示的特征数量

(会剔除空值)

savePath:str 图片存储路径

hyperparams: alpha(alpharange), cv, tol

"""

dftemp = df\_input[x\_columns + [y\_column]].dropna()

x = dftemp[x\_columns]

y = dftemp[y\_column]

if searching:

searcher = RandSearcherCV('classification', SVC())

model = searcher(x, y) # ; searcher.report()

else:

model = SVC().fit(x, y)

param\_dict = model.get\_params()

str\_result = '采用支持向量机分类算法进行变量重要度分析，模型参数为:\n' + dic2str(param\_dict, model.\_\_class\_\_.\_\_name\_\_)

weight\_im = abs(permutation\_importance(model, x, y, n\_repeats=10, random\_state=0).importances\_mean)

weight\_im = weight\_im / sum(weight\_im)

df\_result = pd.DataFrame({

'Variable': x\_columns,

'Weight Importance': weight\_im

}).sort\_values(by="Weight Importance", ascending=False)

top\_list = list(df\_result.head(top\_features)["Variable"])

str\_result += '\n重要度最高的{0}个变量（由高到低）分别为：{1}。\n'.format(top\_features, str(top\_list)[1:-1])

plot\_name\_dict = {}

if savePath is not None:

plot\_name\_dict = x5.horizontal\_bar\_plot(

df\_result.head(top\_features).sort\_values(by="Weight Importance", ascending=True),

'Variable',

'Weight Importance',

'Feature Importance (Coefficient)',

savePath,

dpi=dpi,

picFormat=picFormat,

)

return df\_result, str\_result, plot\_name\_dict['pics'], plot\_name\_dict['save\_pics']

##KNN

def \_kneighb\_cfeatures\_importance(

df\_input,

x\_columns,

y\_column,

top\_features,

searching=True,

savePath=None,

dpi=600,

picFormat='jpeg',

):

"""

df\_input:Dataframe

x\_columns:自变量list

y\_column：因变量str

num\_features:图表中展示的特征数量

(会剔除空值)

savePath:str 图片存储路径

hyperparams: alpha(alpharange), cv, tol

"""

dftemp = df\_input[x\_columns + [y\_column]].dropna()

x = dftemp[x\_columns]

y = dftemp[y\_column]

if searching:

searcher = RandSearcherCV('classification', KNeighborsClassifier())

model = searcher(x, y) # ; searcher.report()

else:

model = KNeighborsClassifier().fit(x, y)

param\_dict = model.get\_params()

str\_result = '采用K近邻分类算法进行变量重要度分析，模型参数为:\n' + dic2str(param\_dict, model.\_\_class\_\_.\_\_name\_\_)

weight\_im = abs(permutation\_importance(model, x, y, n\_repeats=10, random\_state=0).importances\_mean)

weight\_im = weight\_im / sum(weight\_im)

df\_result = pd.DataFrame({

'Variable': x\_columns,

'Weight Importance': weight\_im

}).sort\_values(by="Weight Importance", ascending=False)

top\_list = list(df\_result.head(top\_features)["Variable"])

str\_result += '\n重要度最高的{0}个变量（由高到低）分别为：{1}。\n'.format(top\_features, str(top\_list)[1:-1])

plot\_name\_dict = {}

if savePath is not None:

plot\_name\_dict = x5.horizontal\_bar\_plot(

df\_result.head(top\_features).sort\_values(by="Weight Importance", ascending=True),

'Variable',

'Weight Importance',

'Feature Importance (Coefficient)',

savePath,

dpi=dpi,

picFormat=picFormat,

)

return df\_result, str\_result, plot\_name\_dict['pics'], plot\_name\_dict['save\_pics']