import warnings

warnings.filterwarnings("ignore")

import matplotlib.pyplot as plt

plt.rcParams.update({'figure.max\_open\_warning': 0})

import seaborn as sns

from fancyimpute import IterativeImputer

# modelling

import pandas as pd

import numpy as np

from scipy import stats

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

from sklearn.model\_selection import GridSearchCV, RepeatedKFold, cross\_val\_score,cross\_val\_predict,KFold

from sklearn.metrics import make\_scorer,mean\_squared\_error

%matplotlib inline

#导入数据

data\_file = "./yanbao.xlsx"

dataset = pd.read\_excel(data\_file)

data = dataset.drop(["等级编码"],axis=1)

# 删除具有缺失值的行并重置索引

data\_clean = data.dropna()

data\_clean.reset\_index(drop=True, inplace=True)

data\_clean.info()

data\_clean.head(15)

class\_counts\_all = data\_clean["岩爆等级编码"].value\_counts()

print(class\_counts\_all)

#归一化处理

from sklearn.preprocessing import MinMaxScaler

# 选择需要归一化的特征列，排除"Pillar Stability"列

feature\_cols = data\_clean.columns.drop('岩爆等级编码')

# 创建归一化对象

scaler = MinMaxScaler()

# 创建一个新的 DataFrame 来存放归一化的数据

normalized\_dataset = data\_clean.copy()

# 对特征列进行归一化

normalized\_dataset[feature\_cols] = scaler.fit\_transform(normalized\_dataset[feature\_cols])

#设置特征和目标变量

y = normalized\_dataset["岩爆等级编码"]

X = normalized\_dataset.drop(["岩爆等级编码"],axis=1)

#进行分层抽样划分训练集和测试集

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, stratify=y, random\_state=0)

#将划分好的变量存储到数据集中

train\_data = pd.concat([X\_train, y\_train], axis=1)

test\_data = pd.concat([X\_test, y\_test], axis=1)

# 重置索引，将行索引从1开始，并删除原始索引列

train\_data.reset\_index(drop=True, inplace=True)

test\_data.reset\_index(drop=True, inplace=True)

#复制训练集，并更改特征名称

copy\_train\_data = train\_data.copy()

# 创建一个字典，将旧特征名称映射到新特征名称

column\_mapping1 = {

"H/m": "H/m1",

"σθ/Mpa": "σθ/Mpa1",

"σc/Mpa": "σc/Mpa1",

"σt/Mpa": "σt/Mpa1",

"σθ/σc": "σθ/σc1",

"σc/σt": "σc/σt1",

"Wet": "Wet1",

}

# 重命名特征名称

train\_data = train\_data.rename(columns=column\_mapping1)

# 创建一个字典，将旧特征名称映射到新特征名称

column\_mapping2 = {

"H/m": "H/m2",

"σθ/Mpa": "σθ/Mpa2",

"σc/Mpa": "σc/Mpa2",

"σt/Mpa": "σt/Mpa2",

"σθ/σc": "σθ/σc2",

"σc/σt": "σc/σt2",

"Wet": "Wet2",

}

# 重命名特征名称

copy\_train\_data = copy\_train\_data.rename(columns=column\_mapping2)

# 查看更新后的数据集

copy\_train\_data.head(5)

import pandas as pd

from sklearn.utils import resample

# 假设你的数据集是一个DataFrame，列名为'category'的列是类别标签

df = train\_data

# 计算每个类别的样本数量

class\_counts = df['岩爆等级编码'].value\_counts()

# 找出样本数量最少的类别和它的数量

min\_class = class\_counts.idxmin()

min\_count = class\_counts.min()

# 对于每个类别进行欠采样

resampled\_df = pd.DataFrame()

for category in df['岩爆等级编码'].unique():

category\_df = df[df['岩爆等级编码'] == category]

if category != min\_class:

# 对于样本数量多的类别进行随机抽样

category\_df = resample(category\_df, replace=False, n\_samples=min\_count, random\_state=123)

resampled\_df = pd.concat([resampled\_df, category\_df], axis=0)

# 现在，resampled\_df就是经过欠采样后的数据集

resampled\_df.reset\_index(drop=True, inplace=True)

resampled\_df.head(5)

#使用少样本方法进行训练集的融合扩充

combined\_train\_data = pd.DataFrame()

for index2, row2 in copy\_train\_data.iterrows():

# 遍历 df1 的行

for index1, row1 in train\_data.iterrows():

# 将 df2 的当前行与 df1 的每一行横向组合（排除 "target" 列）

combined\_row = pd.concat([row1.drop("岩爆等级编码"), row2.drop("岩爆等级编码")])

# 根据两个特征数据的目标变量是否相同设置新目标变量的值

new\_target = 1 if row1["岩爆等级编码"] == row2["岩爆等级编码"] else 0

# 将新目标变量添加到组合后的行

combined\_row['新编码'] = new\_target

# 将组合后的行添加到 combined\_df

combined\_train\_data = combined\_train\_data.append(combined\_row, ignore\_index=True)

# 设置新数据集的列名

combined\_train\_data.columns = train\_data.drop("岩爆等级编码", axis=1).columns.tolist() + copy\_train\_data.drop("岩爆等级编码", axis=1).columns.tolist() + ['新编码']

combined\_train\_data.info()

combined\_train\_data.head(5)

# 计算每个类别的样本数量

class\_counts = combined\_train\_data['新编码'].value\_counts()

# 找出样本数量最少的类别和它的数量

min\_class = class\_counts.idxmin()

min\_count = class\_counts.min()

# 对于每个类别进行欠采样

resampled\_combined\_train\_data = pd.DataFrame()

for category in combined\_train\_data['新编码'].unique():

category\_combined\_train\_data = combined\_train\_data[combined\_train\_data['新编码'] == category]

if category != min\_class:

# 对于样本数量多的类别进行随机抽样

category\_combined\_train\_data = resample(category\_combined\_train\_data, replace=False, n\_samples=min\_count, random\_state=123)

resampled\_combined\_train\_data = pd.concat([resampled\_combined\_train\_data, category\_combined\_train\_data], axis=0)

# 现在，resampled\_df就是经过欠采样后的数据集

resampled\_combined\_train\_data.reset\_index(drop=True, inplace=True)

resampled\_combined\_train\_data.head(5)

# 重命名特征名称

test\_data = test\_data.rename(columns=column\_mapping2)

# 查看更新后的数据集

test\_data.head(5)

#将验证集和测试集组合为融合测试集的特征（包含融合特征、训练集原有标签列和测试集编号）

# 初始化一个空的 DataFrame 用于存储新的数据集

combined\_test\_data = pd.DataFrame()

# 遍历数据集1的每一行

for index1, row1 in resampled\_df.iterrows():

# 遍历数据集2的每一行

for index2, row2 in test\_data.iterrows():

# 横向组合特征

combined\_row = pd.concat([row1, row2.drop('岩爆等级编码')])

# 计算新列的值

new\_column\_value = index2

new\_target = 1 if row1["岩爆等级编码"] == row2["岩爆等级编码"] else 0

# 将新列和新目标变量添加到组合行中

combined\_row['数据编号'] = new\_column\_value

combined\_row['新编码'] = new\_target

# 将组合行添加到新数据集中

combined\_test\_data = combined\_test\_data.append(combined\_row,ignore\_index=True)

# 显示数据信息以及前五行内容

combined\_test\_data.info()

combined\_test\_data.head(5)

#定义训练集和测试集的特征和标签

combined\_X\_train = resampled\_combined\_train\_data.drop(["新编码"],axis=1)

combined\_X\_test = combined\_test\_data.drop(["岩爆等级编码", "数据编号","新编码"],axis=1)

combined\_y\_train = resampled\_combined\_train\_data["新编码"]

combined\_y\_test = combined\_test\_data["新编码"]

import numpy as np

import xgboost as xgb

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

from sklearn.metrics import make\_scorer, f1\_score

from bayes\_opt import BayesianOptimization

import matplotlib.pyplot as plt

clf = xgb.XGBClassifier(objective='binary:logistic', random\_state=2017)

def xgb\_evaluate(max\_depth, gamma, min\_child\_weight, subsample, colsample\_bytree, reg\_alpha, reg\_lambda):

params = {

'max\_depth': int(max\_depth),

'gamma': gamma,

'min\_child\_weight': min\_child\_weight,

'subsample': subsample,

'colsample\_bytree': colsample\_bytree,

'reg\_alpha': reg\_alpha,

'reg\_lambda': reg\_lambda,

}

clf.set\_params(\*\*params)

scorer = make\_scorer(f1\_score, average='binary')

cv = StratifiedKFold(n\_splits=5, shuffle=True, random\_state=0)

val\_scores = []

for train\_index, val\_index in cv.split(combined\_X\_train, combined\_y\_train):

X\_train\_fold, y\_train\_fold = combined\_X\_train.iloc[train\_index], combined\_y\_train.iloc[train\_index]

X\_val\_fold, y\_val\_fold = combined\_X\_train.iloc[val\_index], combined\_y\_train.iloc[val\_index]

# Added early stopping

clf.fit(X\_train\_fold, y\_train\_fold, early\_stopping\_rounds=10, eval\_set=[(X\_val\_fold, y\_val\_fold)], verbose=False)

val\_scores.append(f1\_score(y\_val\_fold, clf.predict(X\_val\_fold), average='binary'))

return np.mean(val\_scores)

param\_bounds = {

'max\_depth': (3, 8), # decrease max\_depth

'gamma': (1, 2), # increase gamma

'min\_child\_weight': (8, 10), # increase min\_child\_weight

'subsample': (0.5, 1),

'colsample\_bytree': (0.5, 1),

'reg\_alpha': (3, 5), # add regularization parameter

'reg\_lambda': (2, 8), # add regularization parameter

}

optimizer = BayesianOptimization(f=xgb\_evaluate, pbounds=param\_bounds, random\_state=0, verbose=2)

optimizer.maximize(init\_points=10, n\_iter=60)

print("Best parameters found: ", optimizer.max['params'])

print("Best f1 score: ", optimizer.max['target'])

best\_params = optimizer.max['params']

best\_params['max\_depth'] = int(best\_params['max\_depth'])

best\_clf = xgb.XGBClassifier(objective='binary:logistic', random\_state=2017, \*\*best\_params)

# Added ea9\*+6

best\_clf.fit(combined\_X\_train, combined\_y\_train, early\_stopping\_rounds=10, eval\_set=[(combined\_X\_test, combined\_y\_test)], verbose=False)

test\_f1 = f1\_score(combined\_y\_test, best\_clf.predict(combined\_X\_test), average='binary')

combined\_y\_pred = best\_clf.predict(combined\_X\_test)

print("Test f1 score: ", test\_f1)

#将预测出的新编码与测试集特征相互组合

# 假设 X\_test 是一个 DataFrame，其中包含测试集的特征数据

# 假设 y\_pred 是一个 NumPy 数组或列表，其中包含预测的标签

# 将 y\_pred 转换为 DataFrame

combined\_y\_pred = pd.DataFrame(combined\_y\_pred, columns=["新编码"])

combined\_test1 = combined\_test\_data.drop(["新编码"],axis=1)

# 水平拼接（横向组合）X\_test 和 y\_pred\_df

combined\_pred\_data = pd.concat([combined\_test1.reset\_index(drop=True), combined\_y\_pred], axis=1)

# 显示组合后的 DataFrame

combined\_pred\_data.info()

combined\_pred\_data.head(5)

# 假设你有一个名为data的DataFrame，列名为"column\_name"

column\_name = "新编码"

# 使用value\_counts()方法获取各类别的数量

category\_counts = combined\_pred\_data[column\_name].value\_counts()

# 打印结果

print(f"Category counts in '{column\_name}':\n{category\_counts}")

#根据融合后的数据集进行测试集标签的最终预测

import pandas as pd

from collections import Counter

# 假设你已经有了一个数据集，我们称之为data

# 假设"数据编号"、"新编码"和"岩爆等级编码"是data中的列名

# 获取所有唯一的数据编号

unique\_ids = combined\_pred\_data['数据编号'].unique()

# 创建一个空的 DataFrame 用于存储结果

y\_pred = pd.DataFrame(columns=['y\_pred'], index=unique\_ids)

# 遍历每一个唯一的数据编号

for data\_id in unique\_ids:

# 获取当前数据编号对应的所有行

current\_data = combined\_pred\_data[combined\_pred\_data['数据编号'] == data\_id]

# 选择"新编码"列中值为1的数据

selected\_data = current\_data[current\_data['新编码'] == 1]

# 计算"岩爆等级编码"列中出现的类别及其频率

class\_counter = Counter(selected\_data['岩爆等级编码'])

# 选出频率最高的类别作为最终类别

if len(class\_counter) > 0:

most\_common\_class = class\_counter.most\_common(1)[0][0]

y\_pred.loc[data\_id, 'y\_pred'] = most\_common\_class

else:

print(f"No valid data found for data\_id {data\_id}.")

y\_pred.loc[data\_id, 'y\_pred'] = 4 # 或者设置为一个默认值，如 -1

# 显示 y\_pred 的前几行

y\_pred.head(5)

#查看预测值与真实值之间的偏差

from sklearn.metrics import confusion\_matrix, accuracy\_score, precision\_score, recall\_score, f1\_score, classification\_report

y\_true = test\_data["岩爆等级编码"].tolist()

y\_pred1 = y\_pred['y\_pred'].tolist()

# 创建一个包含新类别的标签列表（浮点数）

labels = [0.0, 1.0, 2.0, 3.0,]

# 计算混淆矩阵

cm = confusion\_matrix(y\_true, y\_pred1, labels=labels)

print("Confusion Matrix:\n", cm)

# 计算准确率

accuracy = accuracy\_score(y\_true, y\_pred1)

print("Accuracy:", accuracy)

# 计算精确度

precision = precision\_score(y\_true, y\_pred1, labels=labels, average='weighted', zero\_division=1)

print("Precision:", precision)

# 计算召回率

recall = recall\_score(y\_true, y\_pred1, labels=labels, average='weighted', zero\_division=1)

print("Recall:", recall)

# 计算F1分数

f1 = f1\_score(y\_true, y\_pred1, labels=labels, average='weighted', zero\_division=1)

print("F1 Score:", f1)

# 生成分类报告

report = classification\_report(y\_true, y\_pred1, labels=labels, zero\_division=1)

print("Classification Report:\n", report)