from autogluon.tabular import TabularDataset, TabularPredictor  
import pandas as pd  
import numpy as np

# 训练集和测试集文件路径  
train\_path = r'D:\xjtufiles\3ee\DGA\datasets\DGA\_data\_add\_cleaned\_train.csv'  
test\_path = r'D:\xjtufiles\3ee\DGA\datasets\DGA\_data\_add\_cleaned\_test.csv'  
  
# 预处理训练集  
df\_train = pd.read\_csv(train\_path)  
  
df\_train['h2'] = pd.to\_numeric(df\_train['h2'], errors='coerce')  
df\_train['ch4'] = pd.to\_numeric(df\_train['ch4'], errors='coerce')  
df\_train['c2h6'] = pd.to\_numeric(df\_train['c2h6'], errors='coerce')  
df\_train['c2h4'] = pd.to\_numeric(df\_train['c2h4'], errors='coerce')  
df\_train['c2h2'] = pd.to\_numeric(df\_train['c2h2'], errors='coerce')  
df\_train['act'] = pd.to\_numeric(df\_train['act'], errors='coerce')  
  
df\_train = df\_train.reset\_index(drop=True) # 重置索引  
  
# 预处理测试集  
df\_test = pd.read\_csv(test\_path)  
  
df\_test['h2'] = pd.to\_numeric(df\_test['h2'], errors='coerce')  
df\_test['ch4'] = pd.to\_numeric(df\_test['ch4'], errors='coerce')  
df\_test['c2h6'] = pd.to\_numeric(df\_test['c2h6'], errors='coerce')  
df\_test['c2h4'] = pd.to\_numeric(df\_test['c2h4'], errors='coerce')  
df\_test['c2h2'] = pd.to\_numeric(df\_test['c2h2'], errors='coerce')  
df\_test['act'] = pd.to\_numeric(df\_test['act'], errors='coerce')  
  
df\_test = df\_test.reset\_index(drop=True) # 重置索引  
  
# 现在，df\_train 和 df\_test 已经分别完成了预处理  
print("训练集预处理完成，形状：", df\_train.shape)  
print("测试集预处理完成，形状：", df\_test.shape)

feature\_cols = ['h2', 'ch4', 'c2h6', 'c2h4', 'c2h2']  
# 确保特征列为数值类型 (df\_train)  
for col in feature\_cols:  
 df\_train[col] = pd.to\_numeric(df\_train[col], errors='coerce')  
  
# 确保目标列为整数类型 (df\_train)  
df\_train['act'] = df\_train['act'].astype(int)  
  
print("训练集数据类型：")  
print(df\_train.dtypes)  
  
# 确保特征列为数值类型 (df\_test)  
for col in feature\_cols:  
 df\_test[col] = pd.to\_numeric(df\_test[col], errors='coerce')  
  
# 确保目标列为整数类型 (df\_test)  
df\_test['act'] = df\_test['act'].astype(int)  
  
print("\n测试集数据类型：")  
print(df\_test.dtypes)

import pandas as pd  
import numpy as np  
  
# 假设 X 是一个 Pandas DataFrame，包含 h2, ch4, c2h6, c2h4, c2h2 列  
  
train\_all = df\_train.copy()  
test\_all = df\_test.copy()  
  
def all\_features(X):  
   
  
 R1 = X['c2h2'] / (X['c2h2'] + X['ch4'] + X['c2h6'])  
 # R2= X['ch4'] / (X['c2h2'] + X['ch4'] + X['c2h6'])  
 # R3 = X['c2h6'] / (X['c2h2'] + X['ch4'] + X['c2h6'])  
 R4 = X['c2h4'] / (X['c2h6'])  
 R5 = X['c2h2'] / (X['c2h4'])  
 R6 = X['c2h2'] / (X['h2'] + X['ch4'] + X['c2h2'] + X['c2h4'] + X['c2h6'])  
 R7 = X['c2h2'] / (X['ch4'])  
 R8 = (X['c2h2']+X['ch4']) / (X['h2'] )  
 # R9 = X['h2'] / (X['c2h2'])  
 # R10 = X['c2h2'] / (X['c2h4'] + X['ch4'] + X['c2h6'])  
 # R11 = (X['ch4'] + X['c2h6']) / (X['h2'] + X['ch4'] + X['c2h2'] + X['c2h4'] + X['c2h6'])  
 # E\_H = X['energy\_index'] = (X['h2'] \* 1 + X['ch4'] \* 2 + X['c2h6'] \* 3 + X['c2h4'] \* 4 + X['c2h2'] \* 5)  
 # # E\_L = ( X['h2'] \* 1 + X['ch4'] \* 2 + X['c2h6'] \* 3)  
 E = (X['h2'] \* 1 + X['ch4'] \* 2 + X['c2h6'] \* 3 + X['c2h4'] \* 4 + X['c2h2'] \* 5)  
 # 处理异常值  
 R1 = R1.replace([np.inf, -np.inf], 0).fillna(0)  
 R4 = R4.replace([np.inf, -np.inf], 0).fillna(0)  
 R5 = R5.replace([np.inf, -np.inf], 0).fillna(0)  
 R6 = R6.replace([np.inf, -np.inf], 0).fillna(0)  
 R7 = R7.replace([np.inf, -np.inf], 0).fillna(0)  
 R8 = R8.replace([np.inf, -np.inf], 0).fillna(0)  
   
 E = E.replace([np.inf, -np.inf], 0).fillna(0)  
 # 添加到特征矩阵  
 X['R1: C2H2/(C2H2+CH4+C2H6)'] = R1  
 X['R4: C2H4/C2H6'] = R4  
 X['R5: C2H2/C2H4'] = R5  
 X['R6: C2H2/(H2+CH4+C2H2+C2H4+C2H6)'] = R6  
 X['R7: C2H2/CH4'] = R7  
 X['R8: (C2H2+CH4)/H2'] = R8  
  
 return X  
  
all\_train\_data = all\_features(train\_all)  
all\_test\_data = all\_features(test\_all)

# Gini随机森林 + SMOTE

import matplotlib.pyplot as plt  
import seaborn as sns  
from collections import OrderedDict  
from sklearn.model\_selection import train\_test\_split  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.metrics import (accuracy\_score, precision\_score, recall\_score,  
 f1\_score, classification\_report, confusion\_matrix)  
from imblearn.over\_sampling import SMOTE, ADASYN, BorderlineSMOTE, KMeansSMOTE  
  
X\_train = all\_train\_data.drop('act', axis=1)  
y\_train = all\_train\_data['act']  
X\_test = all\_test\_data.drop('act', axis=1)  
y\_test = all\_test\_data['act']  
  
# 标签映射（可选）  
label\_mapping = {  
 1: 'HED',  
 2: 'HT',  
 3: 'LED',  
 4: 'LT',  
 5: 'MT',  
 6: 'PD'  
}  
target\_names = [label\_mapping[i] for i in sorted(label\_mapping.keys())]  
  
# 修改 train\_and\_evaluate 函数，返回分类报告和 overall metrics  
def train\_and\_evaluate(X\_train, y\_train, X\_test, y\_test):  
 model = RandomForestClassifier(n\_estimators=100, criterion='gini', random\_state=42)  
 model.fit(X\_train, y\_train)  
 y\_pred = model.predict(X\_test)  
  
 # 每类指标  
 report\_dict = classification\_report(y\_test, y\_pred, target\_names=target\_names, output\_dict=True, zero\_division=0)  
   
 # 转换为DataFrame方便展示  
 report\_df = pd.DataFrame(report\_dict).transpose()  
  
 return report\_df  
  
# 存储结果  
results = OrderedDict()  
results['No Oversampling'] = train\_and\_evaluate(X\_train, y\_train, X\_test, y\_test)  
  
# 只启用 SMOTE（你可以取消注释其他方法）  
resamplers = {  
 'SMOTE': SMOTE(random\_state=42),  
 # 'ADASYN': ADASYN(random\_state=42),  
 # 'BorderlineSMOTE': BorderlineSMOTE(random\_state=42),  
 # 'KMeansSMOTE': KMeansSMOTE(random\_state=42)  
}  
  
for name, sampler in resamplers.items():  
 X\_resampled, y\_resampled = sampler.fit\_resample(X\_train, y\_train)  
 results[name] = train\_and\_evaluate(X\_resampled, y\_resampled, X\_test, y\_test)  
  
# 输出结果  
print("\n=== 每类指标报告 ===")  
for method, report\_df in results.items():  
 print(f"\n[{method}]")  
 print(report\_df.loc[target\_names][['precision', 'recall', 'f1-score']].round(4))

# Gini随机森林 + SMOTE + SHAP

import shap  
  
# 应用 SMOTE  
smote = SMOTE(random\_state=42)  
X\_resampled, y\_resampled = smote.fit\_resample(X\_train, y\_train)  
  
# 训练模型  
model = RandomForestClassifier(n\_estimators=100, criterion='gini', random\_state=42)  
model.fit(X\_resampled, y\_resampled)  
  
# 模型预测并评估  
y\_pred = model.predict(X\_test)  
explainer = shap.TreeExplainer(model)  
shap\_values = explainer.shap\_values(X\_test)  
  
for i, class\_name in enumerate(model.classes\_):  
 #plt.title(f"SHAP Summary Plot for Class {class\_name}", fontsize=15)  
 shap.summary\_plot(shap\_values[:, :, i], X\_test, class\_names=[class\_name], title=f"SHAP Summary Plot for Class {class\_name}")  
   
 plt.show()