from autogluon.tabular import TabularDataset, TabularPredictor  
import pandas as pd  
import numpy as np  
  
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  
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  
  
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  
  
feature\_cols = ['h2', 'ch4', 'c2h6', 'c2h4', 'c2h2']  
  
for col in feature\_cols:  
 df\_train[col] = pd.to\_numeric(df\_train[col], errors='coerce')  
 df\_test[col] = pd.to\_numeric(df\_test[col], errors='coerce')  
  
df\_train['act'] = df\_train['act'].astype(int)  
df\_test['act'] = df\_test['act'].astype(int)

train\_Duval = df\_train.copy()  
test\_Duval = df\_test.copy()  
  
def Duval\_triangle\_features(X):  
  
 R1 = X['c2h2'] / (X['c2h2'] + X['ch4'] + X['c2h6']+ 1e-6 )  
 R2 = X['ch4'] / (X['c2h2'] + X['ch4'] + X['c2h6'] + 1e-6)  
 R3 = X['c2h6'] / (X['c2h2'] + X['ch4'] + X['c2h6']+ 1e-6 )  
  
 # 处理异常值  
 R1 = R1.replace([np.inf, -np.inf], 0).fillna(0)  
 R2 = R2.replace([np.inf, -np.inf], 0).fillna(0)  
 R3 = R3.replace([np.inf, -np.inf], 0).fillna(0)  
  
 X['R1: %C2H2'] = R1  
 X['R2: %CH4'] = R2  
 X['R3: %C2H6'] = R3  
  
 return X  
  
Duval\_train\_data = Duval\_triangle\_features(train\_Duval)  
Duval\_test\_data = Duval\_triangle\_features(test\_Duval)

## 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  
  
# 标签映射  
category\_mapping = {  
 1: 'HED',  
 2: 'HT',  
 3: 'LED',  
 4: 'LT',  
 5: 'MT',  
 6: 'PD'  
}  
target\_names = [category\_mapping[i] for i in sorted(category\_mapping.keys())]  
  
# 提取特征和标签  
X\_train = Duval\_train\_data.drop('act', axis=1)  
y\_train = Duval\_train\_data['act']  
X\_test = Duval\_test\_data.drop('act', axis=1)  
y\_test = Duval\_test\_data['act']  
  
# 模型训练与评估函数  
def train\_and\_evaluate(X\_train, y\_train, X\_test, y\_test, return\_pred=False):  
 model = RandomForestClassifier(n\_estimators=100, criterion='gini', random\_state=42)  
 model.fit(X\_train, y\_train)  
 y\_pred = model.predict(X\_test)  
  
 # 总体指标  
 overall\_metrics = {  
 'Accuracy': accuracy\_score(y\_test, y\_pred),  
 'Precision': precision\_score(y\_test, y\_pred, average='macro', zero\_division=0),  
 'Recall': recall\_score(y\_test, y\_pred, average='macro', zero\_division=0),  
 'F1': f1\_score(y\_test, y\_pred, average='macro', zero\_division=0)  
 }  
  
 # 每类指标报告  
 report\_df = pd.DataFrame(classification\_report(y\_test, y\_pred, target\_names=target\_names, output\_dict=True, zero\_division=0)).transpose()  
  
 if return\_pred:  
 return overall\_metrics, report\_df, y\_pred, model  
 else:  
 return overall\_metrics, report\_df  
# 重采样方法  
resamplers = {  
 'No Oversampling': None,  
 'SMOTE': SMOTE(random\_state=42),  
 # 'ADASYN': ADASYN(random\_state=42),  
 # 'BorderlineSMOTE': BorderlineSMOTE(random\_state=42),  
 # 'KMeansSMOTE': KMeansSMOTE(random\_state=42)  
}  
  
results = OrderedDict()  
  
for method, sampler in resamplers.items():  
 if sampler is None:  
 X\_res, y\_res = X\_train, y\_train  
 else:  
 X\_res, y\_res = sampler.fit\_resample(X\_train, y\_train)  
  
 if method == "SMOTE":  
 overall, report\_df, y\_pred, model = train\_and\_evaluate(X\_res, y\_res, X\_test, y\_test, return\_pred=True)  
 # 混淆矩阵可视化  
 plt.figure(figsize=(8, 6))  
 cm = confusion\_matrix(y\_test, y\_pred, labels=sorted(model.classes\_))  
 class\_names = [category\_mapping[i] for i in sorted(model.classes\_)]  
 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',  
 xticklabels=class\_names, yticklabels=class\_names,  
 annot\_kws={"size": 18, "fontname": "Times New Roman"})  
 plt.xlabel('Predicted Label', fontsize=18, fontname='Times New Roman')  
 plt.ylabel('True Label', fontsize=18, fontname='Times New Roman')  
 plt.title(f'Confusion Matrix - {method}', fontsize=18, fontname='Times New Roman')  
 plt.xticks(fontsize=14, fontname='Times New Roman')  
 plt.yticks(fontsize=14, fontname='Times New Roman')  
 plt.tight\_layout()  
 plt.show()  
 else:  
 overall, report\_df = train\_and\_evaluate(X\_res, y\_res, X\_test, y\_test)  
  
 results[method] = (overall, report\_df)  
  
print("\n=== 模型总体评估指标 ===")  
for method, (metrics, \_) in results.items():  
 print(f"\n[{method}]")  
 for metric, value in metrics.items():  
 print(f"{metric}: {value:.4f}")  
  
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  
  
smote = SMOTE(random\_state=42)  
X\_resampled, y\_resampled = smote.fit\_resample(X\_train, y\_train)  
  
model = RandomForestClassifier(n\_estimators=100, criterion='gini', random\_state=42)  
model.fit(X\_resampled, y\_resampled)  
y\_pred = model.predict(X\_test)  
  
explainer = shap.TreeExplainer(model)  
shap\_values = explainer.shap\_values(X\_test)

# 设置全局字体为 Times New Roman  
plt.rcParams["font.family"] = "Times New Roman"  
matplotlib.rc('font', family='Times New Roman') # 兼容设置  
  
for i, class\_name in enumerate(model.classes\_):  
 shap.summary\_plot(shap\_values[:, :, i], X\_test, class\_names=[class\_name],  
 title=f"SHAP Summary Plot for Class {class\_name}")  
 plt.show()

for feature\_name in X\_test.columns:  
 feature\_index = X\_test.columns.get\_loc(feature\_name)  
 mean\_shap\_values = shap\_values[:, feature\_index, :].mean(axis=0)  
  
 class\_labels = [category\_mapping[cls] for cls in model.classes\_]  
  
 plt.figure()  
 plt.bar(class\_labels, mean\_shap\_values)  
 plt.xlabel("Fault Class", fontsize=12)  
 plt.ylabel("Mean SHAP Value", fontsize=12)  
 plt.title(f"Mean SHAP Value for {feature\_name}", fontsize=14)  
 plt.xticks(rotation=30)  
 plt.tight\_layout()  
 plt.show()

model\_colors = [  
 '#72b6a1', '#e99675', '#95a3c3', '#db96c0',  
 '#b0cc5a', '#e2c148', '#dfc7a3', '#bdbdbd'  
]  
  
# 特征名  
feature\_names = X\_test.columns.tolist()  
num\_features = len(feature\_names)  
num\_classes = len(model.classes\_)  
  
# 创建平均 SHAP 值矩阵：shape = [num\_features, num\_classes]  
mean\_shap\_matrix = np.zeros((num\_features, num\_classes))  
for i, feature\_name in enumerate(feature\_names):  
 feature\_index = X\_test.columns.get\_loc(feature\_name)  
 mean\_shap\_matrix[i] = shap\_values[:, feature\_index, :].mean(axis=0)  
  
x = np.arange(num\_classes) # 类别数  
bar\_width = 0.1 # 条形宽度  
  
plt.figure(figsize=(12, 6))  
for i in range(num\_features):  
 plt.bar(x + i \* bar\_width, mean\_shap\_matrix[i],  
 width=bar\_width,  
 color=model\_colors[i % len(model\_colors)],  
 label=feature\_names[i])  
  
plt.xticks(x + bar\_width \* (num\_features - 1) / 2, class\_labels)  
plt.xlabel("Fault Class", fontsize=14)  
plt.ylabel("Mean SHAP Value", fontsize=14)  
plt.title("Mean SHAP Value per Class for Each Feature", fontsize=14)  
plt.legend(title="Feature", bbox\_to\_anchor=(1, 1), loc='upper left', fontsize=14)  
plt.tight\_layout()  
plt.show()