import os

import pandas as pd

import numpy as np

from sklearn.preprocessing import RobustScaler

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

def read\_data(data\_paths, encoding='latin1'):

    """

    读取指定路径下的所有 CSV 文件并合并为一个 DataFrame

    :param data\_paths: 数据文件路径列表

    :param encoding: 文件编码，默认为 'latin1'

    :return: 合并后的数据 DataFrame

    """

    data\_frames = []

    for path in data\_paths:

        try:

            df = pd.read\_csv(path, encoding=encoding)  # 指定编码

            # 检查是否存在类似 'Label' 的列名

            label\_columns = [col for col in df.columns if 'Label' in col or 'label' in col]

            if len(label\_columns) == 0:

                print(f"警告：文件 {path} 中未找到类似 'Label' 的列。")

            elif len(label\_columns) > 1:

                print(f"警告：文件 {path} 中找到多个类似 'Label' 的列：{label\_columns}，将使用第一个。")

            if label\_columns:

                df.rename(columns={label\_columns[0]: 'Label'}, inplace=True)

                # 处理Label列为空的行，这里选择删除

                df = df.dropna(subset=['Label'])

                # 打印当前文件中Label列的唯一值

                print(f"文件 {path} 中Label列的唯一值: {df['Label'].unique()}")

            data\_frames.append(df)

            print(f"成功读取文件: {path}，数据列数: {len(df.columns)}")

        except FileNotFoundError:

            print(f"文件 {path} 未找到，跳过。")

        except UnicodeDecodeError:

            print(f"文件 {path} 解码失败，请检查文件编码。")

    if data\_frames:

        result = pd.concat(data\_frames, ignore\_index=True)

        print(f"合并后的数据列数: {len(result.columns)}")

        return result

    return pd.DataFrame()

def data\_cleaning(data):

    """

    数据清洗，填充缺失值，处理无穷大值和超出范围值，去除所有数据值全部为0的列

    :param data: 原始数据 DataFrame

    :return: 清洗后的数据 DataFrame

    """

    # 打印清洗前Label列的唯一值和数量

    print(f"清洗前Label列的唯一值: {data['Label'].unique()}")

    print(f"清洗前Label列的数量: {data['Label'].count()}")

    # 填充缺失值，这里简单用均值填充数值列

    numeric\_cols = data.select\_dtypes(include=[np.number]).columns

    for col in numeric\_cols:

        data[col] = data[col].fillna(data[col].mean())

    # 处理无穷大值，将无穷大值替换为 NaN 再填充

    data.replace([np.inf, -np.inf], np.nan, inplace=True)

    data = data.ffill()

    # 去除所有数据值全部为0的列

    all\_zero\_cols = data.columns[(data == 0).all()]

    data = data.drop(all\_zero\_cols, axis=1)

    # 打印清洗后Label列的唯一值和数量

    print(f"清洗后Label列的唯一值: {data['Label'].unique()}")

    print(f"清洗后Label列的数量: {data['Label'].count()}")

    return data

def encode\_categorical\_features(data, attack\_type\_mapping):

    """

    对非数值型特征进行编码处理，使用预定义的攻击类型映射

    :param data: 数据 DataFrame

    :param attack\_type\_mapping: 攻击类型映射字典

    :return: 编码后的特征 DataFrame

    """

    if 'Label' in data.columns:

        # 添加处理未知标签的逻辑，将未知标签映射为-1

        data['Label'] = data['Label'].map(lambda x: attack\_type\_mapping.get(x, -1))

    return data

def normalize\_numeric\_features(data):

    """

    对数值型特征进行归一化处理

    :param data: 数据 DataFrame

    :return: 归一化后的特征 DataFrame

    """

    scaler = RobustScaler()

    numeric\_cols = data.select\_dtypes(include=[np.number]).columns

    # 检查 'Label' 列是否存在，如果存在则排除

    if 'Label' in numeric\_cols:

        numeric\_cols = numeric\_cols.drop('Label')

    data[numeric\_cols] = scaler.fit\_transform(data[numeric\_cols])

    return data

def split\_train\_test(data, test\_size=0.1, random\_state=42):

    """

    将数据划分为训练集和测试集，比例为9:1

    :param data: 数据 DataFrame

    :param test\_size: 测试集比例，默认为0.1

    :param random\_state: 随机种子

    :return: 训练集 DataFrame、测试集 DataFrame

    """

    train, test = train\_test\_split(data, test\_size=test\_size, random\_state=random\_state)

    return train, test

def save\_processed\_data(train, test, save\_path):

    """

    保存处理后的数据到 CSV 文件

    :param train: 训练集 DataFrame

    :param test: 测试集 DataFrame

    :param save\_path: 保存路径

    """

    # 检查文件夹是否存在，如果不存在则创建

    if not os.path.exists(save\_path):

        os.makedirs(save\_path)

    try:

        train.to\_csv(os.path.join(save\_path, 'train.csv'), index=False)

        test.to\_csv(os.path.join(save\_path, 'test.csv'), index=False)

    except PermissionError as e:

        print(f"权限错误: {e}。请检查文件夹权限或者关闭占用文件的程序。")

    except Exception as e:

        print(f"保存文件时出现错误: {e}")

def save\_attack\_labels(attack\_type\_mapping, save\_path):

    """

    保存攻击类型映射的文档

    :param attack\_type\_mapping: 攻击类型映射字典

    :param save\_path: 保存路径

    """

    # 检查文件夹是否存在，如果不存在则创建

    if not os.path.exists(save\_path):

        os.makedirs(save\_path)

    mapping\_df = pd.DataFrame(list(attack\_type\_mapping.items()), columns=['Attack Type', 'Label'])

    mapping\_df.to\_csv(os.path.join(save\_path, 'attack\_type\_mapping.csv'), index=False)

def main():

    # 攻击标签到数字编码的映射，只保留指定的攻击类型

    attack\_type\_mapping = {

        "BENIGN": 0,

        "Syn": 1,

        "DoS Hulk": 2,

        "TFTP": 3,

        "PortScan": 4,

        "DrDoS\_NTP": 5,

        "Portmap": 6,

        "LDAP": 7,

        "UDP": 8,

        "UDP-lag": 9,

        "DrDoS\_DNS": 10,

        "MSSQL": 11,

        "DrDoS\_UDP": 12,

        "DrDoS\_MSSQL": 13,

        "NetBIOS": 14,

        "DrDoS\_NetBIOS": 15,

        "DrDoS\_LDAP": 16,

        "DrDoS\_SNMP": 17,

        "DoS GoldenEye": 18,

        "DoS slowloris": 19,

        "DoS Slowhttptest": 20,

        "Bot": 21,

        "UDPLag": 22,

        "WebDDoS": 23,

        "Heartbleed": 24

    }

    data\_paths = [

        r"D:\毕业设计\数据预处理\raw\_data\CIC2017\TrafficLabelling\_\Friday-WorkingHours-Afternoon-PortScan.pcap\_ISCX.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CIC2017\TrafficLabelling\_\Friday-WorkingHours-Morning.pcap\_ISCX.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CIC2017\TrafficLabelling\_\Monday-WorkingHours.pcap\_ISCX.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CIC2017\TrafficLabelling\_\Thursday-WorkingHours-Afternoon-Infilteration.pcap\_ISCX.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CIC2017\TrafficLabelling\_\Thursday-WorkingHours-Morning-WebAttacks.pcap\_ISCX.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CIC2017\TrafficLabelling\_\Tuesday-WorkingHours.pcap\_ISCX.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CIC2017\TrafficLabelling\_\Wednesday-workingHours.pcap\_ISCX.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\BENIGN.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\DrDoS\_DNS.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\DrDoS\_LDAP.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\DrDoS\_MSSQL.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\DrDoS\_NetBIOS.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\DrDoS\_NTP.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\DrDoS\_SNMP.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\DrDoS\_UDP.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\LDAP.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\MSSQL.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\NetBIOS.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\Portmap.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\Syn.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\TFTP.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\UDP.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\UDPLag.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\UDP-lag.csv",

        r"D:\毕业设计\数据预处理\raw\_data\CICDDoS2019\class\_split\WebDDoS.csv",

    ]

    save\_path = r'D:\毕业设计\数据预处理\processed\_data'

    # 读取数据

    data = read\_data(data\_paths)

    if 'Label' not in data.columns:

        print("警告：读取的数据中没有 'Label' 列，请检查数据文件。")

        return

    # 数据清洗

    cleaned\_data = data\_cleaning(data)

    # 编码分类特征

    encoded\_data = encode\_categorical\_features(cleaned\_data, attack\_type\_mapping)

    # 归一化数值特征

    normalized\_data = normalize\_numeric\_features(encoded\_data)

    # 划分训练集和测试集

    train, test = split\_train\_test(normalized\_data, test\_size=0.1)

    # 统计处理完的数据的攻击类型及对应数量

    if 'Label' in normalized\_data.columns:

        attack\_type\_counts = normalized\_data['Label'].value\_counts()

        reverse\_attack\_type\_mapping = {v: k for k, v in attack\_type\_mapping.items()}

        reverse\_attack\_type\_mapping[-1] = "未知类型"

        print("处理后的数据的攻击类型及对应数量：")

        for attack\_type\_code, count in attack\_type\_counts.items():

            attack\_type\_name = reverse\_attack\_type\_mapping.get(attack\_type\_code, "未知类型")

            print(f"{attack\_type\_name}: {count}")

    else:

        print("警告：处理后的数据中没有 'Label' 列。")

    # 保存处理后的数据

    save\_processed\_data(train, test, save\_path)

    # 保存攻击类型映射

    save\_attack\_labels(attack\_type\_mapping, save\_path)

if \_\_name\_\_ == "\_\_main\_\_":

    main()