这里选择 Bert 来处理文本分类问题（但是好像细化颗粒度分类上效果不好）
观察给定的训练集发现，在给定的 label 里面不存在 2-3 的分类，但是由于服务器连不上 hf 或者模型下载时间花费过长，这里选择手动利用词典对于训练集中的 label 改变标签啊。
（免责声明：以下的 keyword 不是本人可以说出来的，是 ai 生成的）

```
import pandas as pd
import torch
from torch.utils.data import Dataset
from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
from transformers import DataCollatorWithPadding
from sklearn.preprocessing import LabelEncoder
import numpy as np
import os
import json
import re

def enhanced_keyword_label(text, original_label):

    """使用扩充关键词库和优化匹配逻辑的标签分类函数"""

    # 如果原标签是 0，直接返回 0

    if original_label == 0:

        return 0

    # 以下代码只对原标签为 1 的样本生效

    # 【1】攻击个人关键词 - 大幅扩充

    personal_attack = [

        # 直接攻击用词

        '你这个人', '你自己', '傻', '蠢', '白痴', '弱智', '智障', '脑残', '废物', '垃圾',

        '蠢货', '笨货', '煞笔', '二货', '呆瓜', '饭桶', '窝囊废', '丑八怪', '恶心', '下三滥',

        '臭不要脸', '不知廉耻', '没脑子', '废柴', '没用的', '猪脑子', '低能儿', '没教养',

        # 骂人常用词

        '傻逼', '神经病', '有病', '有毛病', '滚', '闭嘴', '去死', '烦人', '没救了', '无可救药',
```

```
    '无药可救', '蠢材', '蠢蛋', '愚蠢', '愚昧', '愚笨', '笨蛋', '笨猪', '猪头', '猪脑袋',
    '狗屁', '放屁', '放狗屁', '屁话', '胡说八道', '满口胡言', '无知', '无脑', '没大脑',
    '蠢驴', '蠢猪', '没脸没皮', '厚颜无耻', '不要脸', '脸皮厚', '死皮赖脸', '贱人', '贱货',
    # 带有人称代词的短语
    '你懂个', '你懂什么', '你算什么', '你以为你是谁', '你算老几', '你有病吧', '你有毛病',
    '你脑子有坑', '你脑子有泡', '你脑子有洞', '你脑子进水', '你怎么这么', '你这么蠢',
    '你这种人', '你这种废物', '你活该', '你该死', '你去死', '你去死吧', '你滚', '你滚蛋',
    '你给我闭嘴', '你不配', '你算个屁', '你什么玩意', '你这玩意', '你这货', '你这废物',
    # 带有贬义情感的评价词
    '无能', '无用', '无知', '可悲', '可怜', '可笑', '可恶', '可恨', '讨厌', '烦人',
    '令人作呕', '令人恶心', '令人发指', '令人厌恶', '低级', '低俗', '低劣', '卑劣',
    '卑鄙', '肮脏', '龌龊', '恶心', '恶毒', '恶劣', '可耻', '无耻', '丢人', '丢脸',
]
# 【2】攻击群体关键词 - 大幅扩充
group_attack = [
    # 基础群体攻击词
    '你们这些', '某些人', '群体', '民族', '女人', '男人', '穷鬼', '乡巴佬', '死肥宅',
    '娘娘腔', '臭男人', '臭女人', '矮矬穷', '黑鬼', '白皮猪', '黄皮耗子', '女拳', '男拳',
    '女同', '男同', '一群', '这帮人', '集体', '都是一样', '都这样', '这种人', '整个',
    '劣等', '民族性',
    # 群体标签词
    '农村人', '农民', '屌丝', '穷人', '富人', '暴发户', '土豪', '外地人', '外地佬',
```

```
        '外国人', '老外', '中国人', '日本人', '韩国人', '美国人', '印度人', '黑人', '白人',
        '黄种人', '亚洲人', '欧洲人', '非洲人', '中东人', '拉美人', '穆斯林', '基督徒',
        '犹太人', '佛教徒', '北方人', '南方人', '东北人', '河南人', '上海人', '北京人',
        '广东人', '普通人', '底层人', '低端人口', '打工人', '老人', '年轻人', '小孩', '学生',
        '女性', '男性', '老头', '老太', '妇女', '大妈', '老阿姨', '老头子', '太君', '棒子',
        '鬼子', '洋鬼子', '阿三', '罗刹', '绿绿', '黄俄', '倭寇', '高丽棒子', '三姓家奴',
        # 群体攻击短语
        '这帮', '那些', '一堆', '一批', '满地都是', '到处都是', '这群', '这些人真是',
        '这些人根本', '这种族群', '这种民族', '这种人种', '这类人', '统统都是', '没有一个好东西',
        '都是垃圾', '全是废物', '没一个好东西', '没有一个例外', '没有一个不是', '没素质', '素质差',
        '天生就是', '从小就是', '骨子里就是', '本性如此', '基因决定', '劣等基因', '智商低下',
        '天生愚钝', '先天不足', '种族劣势', '民族劣根性',
        # 带有群体性质的评价
        '素质低下', '国民性', '民族性', '劣根性', '共性', '普遍现象', '普遍问题', '通病',
        '共同缺点', '群体特征', '普遍特点', '种族特点', '民族特色', '天生如此', '与生俱来',
        '根深蒂固', '无法改变', '无可救药', '救不了', '没救了', '全都一个德行', '没区别',
    ]
    # 【3】反偏见关键词 - 大幅扩充
    anti_bias = [
        # 基础反偏见词
        '包容', '平等', '反歧视', '尊重', '倡导', '多元', '一视同仁', '消除偏见', '和谐共处',
```

```
'求同存异', '相互理解', '共同进步', '团结友爱', '互相尊重', '消除歧视', '反对偏见',
# 扩充反歧视词汇
'反对歧视', '抵制歧视', '杜绝歧视', '抗议歧视', '不要歧视', '不应歧视', '不该歧视',
'不能歧视', '不可歧视', '反对偏见', '消除偏见', '摒弃偏见', '摆脱偏见', '打破偏见',
'破除偏见', '解决偏见', '根除偏见', '纠正偏见', '反对傲慢', '杜绝傲慢', '避免傲慢',
# 倡导积极态度的词汇
'相互尊重', '彼此尊重', '互相尊重', '互相理解', '相互理解', '彼此理解', '相互包容',
'彼此包容', '互相包容', '平等对待', '一视同仁', '同等对待', '公平对待', '公正对待',
'以人为本', '以人为尊', '人人平等', '平等权利', '基本权利', '基本人权', '人权保障',
'尊严保障', '平等保障', '人格尊严', '人格平等', '人格独立', '宽容', '宽容对待',
# 反歧视短语
'不分男女', '不分老幼', '不分种族', '不分肤色', '不分民族', '不分地域', '不分贫富',
'不分高低', '不分贵贱', '不分好坏', '不分优劣', '不论出身', '不论背景', '不论家庭',
'不论职业', '不论教育', '不论外表', '不论年龄', '人皆生而平等', '天赋人权', '固有尊严',
'先天平等', '人人生而自由', '平等机会', '同等机会', '公平机会', '公正评价', '客观评价',
# 多元化词汇
'多元文化', '文化多样性', '文明多样性', '种族多样性', '民族多样性', '文化差异',
'民族差异', '个体差异', '求同存异', '互利共赢', '合作共赢', '共同发展', '共同进步',
'和谐发展', '和谐社会', '和谐共处', '和平共处', '友好相处', '兼容并蓄', '海纳百川',
'有容乃大', '接纳差异', '承认差异', '欣赏差异', '保护差异',
]
```

```
# 复杂模式匹配 - 攻击个人
personal_patterns = [
    r'你[是的真]+(真的)?(就是)?(太)?(非常)?[傻蠢笨弱智脑残垃圾废物白痴]',
    r'你.*?(脑子进水|缺根筋|缺条筋|有病|有毛病|神经病)',
    r'你.*?(算什么东西|以为自己是谁|以为你是谁|算老几)',
    r'(滚蛋|滚远点|闭嘴|住嘴|闭嘴|住口)',
    r'(傻逼|二逼|煞笔|傻 x|傻 b|妈的|去死)',
]
# 复杂模式匹配 - 攻击群体
group_patterns = [
    r'(他们|她们|这些人|那些人|一群人|这群|这帮|那帮).*?(都是|全是|全部|一样|没区别)',
    r'([东西南北中]方人|[男女老少]人|[各某]国人).*?(素质|劣根性|天生|基因)',
    r'([东西南北中]方人|[男女老少]人|[各某]国人).*?(没有一个|统统|全都是)',
    r'(素质|道德|能力|智商).*?(低下|低能|堪忧|差劲)',
    r'(先天|天生|本性|基因|骨子里|从小).*?(劣根性|缺陷|问题)',
]
# 复杂模式匹配 - 反偏见
antibias_patterns = [
    r'(不应该|不该|不能|不可以|不允许|不要).*?(歧视|偏见|傲慢|轻视|鄙视)',
    r'(反对|抵制|杜绝|打破|消除).*?(歧视|偏见|傲慢|轻视)',
    r'(提倡|倡导|支持|鼓励).*?(平等|尊重|包容|多元|和谐)',
    r'(应该|需要|必须).*?(尊重|平等|公平|客观).*?(对待|看待|评价)',
```

```
    r'(不分|不论|无论).*?(男女|老幼|贫富|种族|民族|地域|肤色)',
]
# 1. 先检查规则匹配
# 攻击个人规则
for pattern in personal_patterns:
    if re.search(pattern, text):
        return 1
# 攻击群体规则
for pattern in group_patterns:
    if re.search(pattern, text):
        return 2
# 反偏见规则
for pattern in antibias_patterns:
    if re.search(pattern, text):
        return 3
# 2. 然后检查关键词匹配
# 攻击个人模式
for word in personal_attack:
    if word in text:
        return 1
# 攻击群体模式
for word in group_attack:
```

```
            if word in text:
                return 2
        # 反偏见模式
        for word in anti_bias:
            if word in text:
                return 3
        # 如果没有匹配到任何规则，但原标签是 1，则默认为攻击个人
        return 1  # 默认为攻击个人，而不是 0

# 1. 读取数据
print("正在读取数据...")
df_train = pd.read_csv('/root/data/NS-2025-03-data/input_data/train.csv')
df_valid = pd.read_csv('/root/data/NS-2025-03-data/input_data/valid.csv')
df_test = pd.read_csv('/root/data/NS-2025-03-data/input_data/test.csv')

# 使用关键词匹配替代零样本分类
print("正在使用关键词匹配标注训练数据...")
sample_size = min(len(df_train), 1000)
df_train_sample = df_train.sample(sample_size)
# 修复这里：传入两个参数而不是一个
df_train_sample['auto_label'] = df_train_sample.apply(
    lambda row: enhanced_keyword_label(row['TEXT'], row['label']), axis=1
```

```
)


print("正在使用关键词匹配标注验证数据...")
sample_size_valid = min(len(df_valid), 200)
df_valid_sample = df_valid.sample(sample_size_valid)
# 修复这里：传入两个参数而不是一个
df_valid_sample['auto_label'] = df_valid_sample.apply(
    lambda row: enhanced_keyword_label(row['TEXT'], row['label']), axis=1
)
# 对样本数据用 auto_label 标签
df_train_sample['final_label'] = df_train_sample['auto_label']
df_valid_sample['final_label'] = df_valid_sample['auto_label']


# 标签编码
offend_encoder = LabelEncoder()
topic_encoder = LabelEncoder()


# 使用 final_label 作为细粒度冒犯标签
df_valid_sample['细粒度冒犯'] = offend_encoder.fit_transform(df_valid_sample['final_label'])
df_valid_sample['话题'] = topic_encoder.fit_transform(df_valid_sample['topic'])
df_train_sample['细粒度冒犯'] = offend_encoder.transform(df_train_sample['final_label'])
df_train_sample['话题'] = topic_encoder.transform(df_train_sample['topic'])
```

```
# 统计类别数
num_offend = len(offend_encoder.classes_)
num_topic = len(topic_encoder.classes_)

# 2. 加载模型 - 使用本地路径或离线模式
print("正在加载 BERT 模型...")
try:
    # 设置离线模式
    os.environ["TRANSFORMERS_OFFLINE"] = "1"
    tokenizer = BertTokenizer.from_pretrained('bert-base-chinese', local_files_only=True)
    model = BertForSequenceClassification.from_pretrained('bert-base-chinese',
                                                          num_labels=num_offend * num_topic,
                                                          local_files_only=True)
except:
    print("无法加载 BERT 模型，尝试使用 RandomForest 替代...")
    from sklearn.ensemble import RandomForestClassifier
    from sklearn.feature_extraction.text import TfidfVectorizer
    # 使用 TF-IDF 提取文本特征
    vectorizer = TfidfVectorizer(max_features=5000)
    X_train = vectorizer.fit_transform(df_train_sample['TEXT'])
    y_train = df_train_sample['细粒度冒犯'] * num_topic + df_train_sample['话题']
```

```
# 训练随机森林模型
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
# 预测
X_test = vectorizer.transform(df_test['TEXT'])
pred_labels = rf_model.predict(X_test)
# 转换预测结果
offense_labels = pred_labels // num_topic
topic_labels = pred_labels % num_topic
offense_labels = offend_encoder.inverse_transform(offense_labels)
# 话题类别映射
topic_mapping = {0: "gender", 1: "race", 2: "region"}
topic_classes = [topic_mapping[label] for label in topic_labels]
# 创建 JSON 格式结果
result = []
for offense, topic in zip(offense_labels, topic_classes):
    entry = {
        "offensive_detection": {
            "label": int(offense)
        },
        "topic_classification": {
            "label": topic
```

```
            }
        }
        result.append(entry)
    # 保存结果
    with open('prediction_results.json', 'w', encoding='utf-8') as f:
        json.dump(result, f, ensure_ascii=False, indent=4)
    print(f"预测结果已保存到 prediction_results.json，共 {len(result)} 条记录")
    exit(0)

# 3. 数据集定义
class MultiLabelDataset(Dataset):
    def __init__(self, df, tokenizer):
        self.texts = df['TEXT'].tolist()
        self.offend = df['细粒度冒犯'].tolist()
        self.topic = df['话题'].tolist()
        self.tokenizer = tokenizer

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        encoding = self.tokenizer(
```

```
            self.texts[idx],
            truncation=True,
            padding='max_length',
            max_length=128,
            return_tensors='pt'
        )
        item = {key: val.squeeze(0) for key, val in encoding.items()}
        item['labels'] = torch.tensor([self.offend[idx], self.topic[idx]], dtype=torch.long)
        return item

# 4. 数据集和 DataLoader
train_dataset = MultiLabelDataset(df_train_sample, tokenizer)
valid_dataset = MultiLabelDataset(df_valid_sample, tokenizer)
data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

# 5. 自定义 Trainer
class CustomTrainer(Trainer):
    def compute_loss(self, model, inputs, return_outputs=False, **kwargs):
        labels = inputs.pop("labels")
        outputs = model(**inputs)
        loss_fct = torch.nn.CrossEntropyLoss()
        offend_labels = labels[:, 0]
```

```
            topic_labels = labels[:, 1]
            combined_labels = offend_labels * num_topic + topic_labels
            loss = loss_fct(outputs.logits, combined_labels)
            return (loss, outputs) if return_outputs else loss

# 6. 训练参数
training_args = TrainingArguments(
    output_dir='./bert_output',
    per_device_train_batch_size=8,
    num_train_epochs=3,
    logging_steps=10,
    save_steps=999999,
    save_total_limit=1,
    learning_rate=2e-5,
    remove_unused_columns=False,
    eval_steps=500,
    do_eval=True,
)

# 7. 训练
print("开始训练模型...")
trainer = CustomTrainer(
```

```
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=valid_dataset,
        data_collator=data_collator,
        tokenizer=tokenizer,
)
trainer.train()

# 8. 预测
print("正在预测测试数据...")
df_test['细粒度冒犯'] = 0
df_test['话题'] = 0
test_dataset = MultiLabelDataset(df_test, tokenizer)
preds = trainer.predict(test_dataset)
pred_labels = np.argmax(preds.predictions, axis=1)

#  将预测标签转换为原始标签
offense_labels = pred_labels // num_topic
topic_labels = pred_labels % num_topic
offense_labels = offend_encoder.inverse_transform(offense_labels)
```

```
# 话题类别映射
topic_mapping = {0: "gender", 1: "race", 2: "region"}
topic_classes = [topic_mapping[label] for label in topic_labels]

# 创建 JSON 格式结果
result = []
for offense, topic in zip(offense_labels, topic_classes):
    entry = {
        "offensive_detection": {
            "label": int(offense)
        },
        "topic_classification": {
            "label": topic
        }
    }
    result.append(entry)

# 保存为 JSON 文件
print("保存预测结果...")
with open('prediction_results.json', 'w', encoding='utf-8') as f:
    json.dump(result, f, ensure_ascii=False, indent=4)
```

```
print(f"预测结果已保存到 prediction_results.json，共 {len(result)} 条记录")
```