BERT多标签分类模型

简介和功能

1.主要目的：根据已有的景点名称和对应的标签来进行语言模型的训练，期望在输入新的景点名称之后，模型能够生成对应的标签。

2.延申功能：可以针对其他的旅游实体进行训练，达到多标签分类的效果。

运行环境和要求

1. Jupyter Notebook：确保你已经安装了Jupyter Notebook并可以运行它。（建议使用anaconda来配置环境）
2. Python环境：确保你已经安装了Python，并且可以在Jupyter Notebook中使用它。
3. PyTorch库：这段代码使用了PyTorch库进行深度学习模型的训练。
4. Transformers库：这段代码还使用了Transformers库
5. GPU支持。

使用方法

1. 下载代码。
2. 检查完环境是否达到要求（建议有GPU资源，使用CPU会导致训练速度过慢）
3. 根据需要修改输入和输出文件的路径名称，如下图：

```
# 读取训练数据
with open('训练数据路径', 'r', encoding='utf-8') as f:
    train_data = json.load(f)

# 读取验证数据
with open('验证数据路径', 'r', encoding='utf-8') as f:
    validation_data = json.load(f)
```

```
# 将结果保存到json文件
with open('训练集的结果保存路径', 'w', encoding='utf-8') as f:
    json.dump(train_results, f)

with open('验证集的结果保存路径', 'w', encoding='utf-8') as f:
    json.dump(valid_results, f)
```

4. 运行直至所有循环完成。
5. 根据保存的数据进行分析。

输入的数据要求

无论是训练集还是验证集，所有数据都在json文件中，格式如下：

```
[
  {
```

```
    "text": "景点一",
    "label": [
      "标签一",...
    ]
  },
  {
    "text": "景点二",
    "label": [
      "标签二",...
    ]
  },
    ...
]
```

注意事项：训练集的数据量要比验证集大几倍，同时需要确保每个标签都有在两个数据集中出现过（不然可能会报错说向量大小不符）

输出的数据格式

输出的数据长这样：

```
[
  {
    "logits": [
      3.1482012271881104,
      -5.3884077072143555,
      -5.269010543823242,
      -5.077752113342285,
      -5.820712566375732,
      -3.990055799484253,
      -5.205063343048096,
      -5.386012077331543,
      -4.852389812469482
    ],
    "labels": [
      1.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0
    ]
  },
  {
    "logits": [
      -4.153970718383789,
      -5.380253314971924,
      -5.827282905578613,
      -5.651237964630127,
      -5.983510494232178,
      3.8447048664093018,
      -6.137881755828857,
      -5.129909992218018,
      -5.729831218719482
    ],
```

```
    "labels": [
      1.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0,
      0.0
    ]
  },
    ...
]
```

如何使用输出结果：

针对"logits"，可以设置不同的阈值来区分某个标签是否可以判定为positive（设置为1），反之为negative（设置为0）.

最后用处理过的"logits"与"labels"（即正确的标签）来进行对比，以分析模型的效果如何。

主要代码流程分析

模型参数的设置：

```
# 加载Bert预训练模型和分词器
model_name = 'bert-base-chinese'  # 或 'bert-base-uncased'（英文）
tokenizer = BertTokenizer.from_pretrained(model_name)
model = BertForSequenceClassification.from_pretrained(model_name, num_labels=len(all_labels))
```

采用的预训练模型是'bert-base-chinese'。

```
def train(model, train_loader, validation_loader, device, num_epochs, threshold=0):
    optimizer = torch.optim.AdamW(model.parameters(), lr=2e-5)
```

其中num_epoches是循环次数（默认为30），optimizer是优化器。

训练流程：

```
    model.train()
    train_loss = 0.0  #初始化loss值

    train_results = []
    for batch in train_loader:
        optimizer.zero_grad()  #遍历所有参数，清空上次训练的参数梯度

        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['label'].to(device)  #上面三行是将数据输入到训练工具上（GPU）

        outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss  #获得单此训练后的loss值
        logits = outputs.logits  #将训练结果进行保存

        loss.backward()  #逆向传播（计算梯度）
        optimizer.step()  #进行一次优化步骤（基于梯度下降）
```

```
        train_loss += loss.item()  #累加loss
        for i in range(len(logits)):
            result = {
                'logits': logits[i].detach().cpu().numpy().tolist(),
                'labels': labels[i].detach().cpu().numpy().tolist()
            }
            train_results.append(result)  #保存结果
```

验证流程：

```
model.eval()
val_loss = 0.0  #初始化loss值
predictions = []  #初始化空的预测值列表

with torch.no_grad():
    valid_results = []
    for batch in validation_loader:
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['label'].to(device)  #上面三行是将数据输入到训练工具上（GPU）

        outputs = model(input_ids=input_ids, attention_mask=attention_mask, labels=labels)
        loss = outputs.loss  #获得单此训练后的loss值
        logits = outputs.logits  #将训练结果进行保存

        val_loss += loss.item()  #累加loss
        predictions.extend(logits.sigmoid().cpu().numpy()) #记录所有预测结果到列表中
        for i in range(len(logits)):
            result = {
                'logits': logits[i].detach().cpu().numpy().tolist(),
                'labels': labels[i].detach().cpu().numpy().tolist()
            }
            valid_results.append(result)  #保存结果
```

实验

数据来源：

1. 来自客路的约800条已分类的数据。
2. Chatgpt打标的约1000条已分类的数据。

数据特点

这些数据中有657条景点是相同的（标签可能会不同），146条是客路的数据（简称为原始数据）独有的，423条是gpt数据独有的。

训练集与验证集的生成策略

首先针对每种标签（9个），计算含有这个标签的总数居，然后按照4：1：1的比例分成训练集和不同的验证集，其中对于每个验证集，优先从自己的数据里提取，如果不够则从另外的数据中提取。

数据结果:

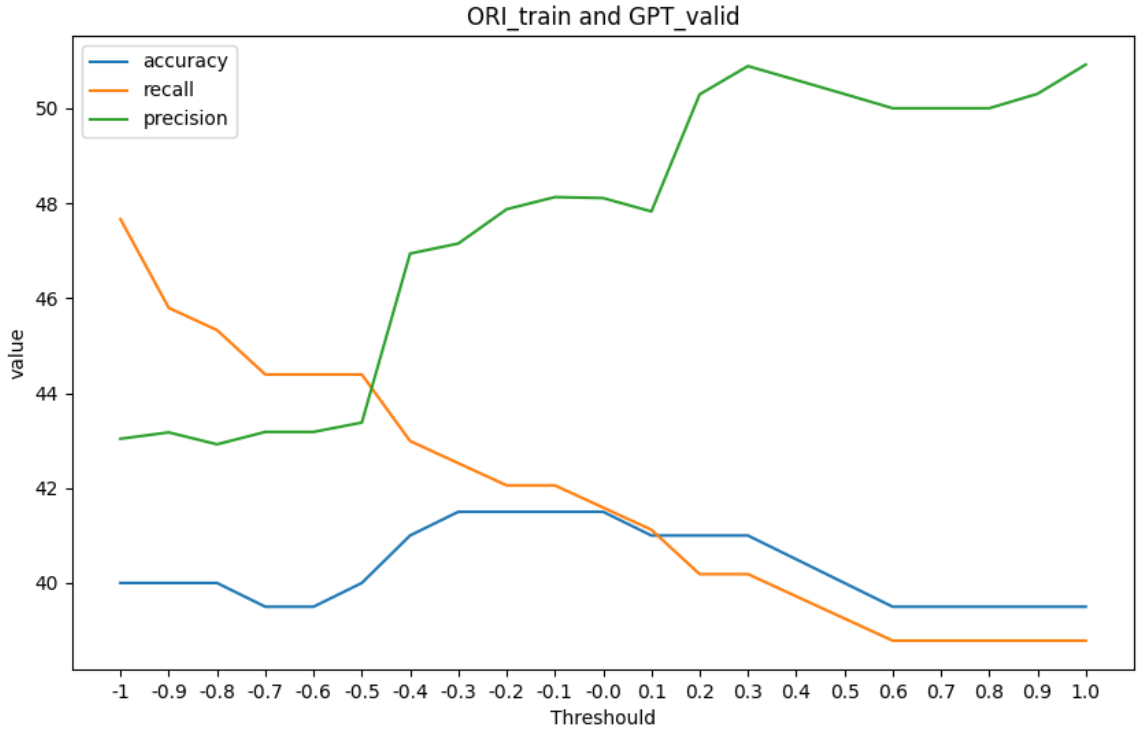
描述

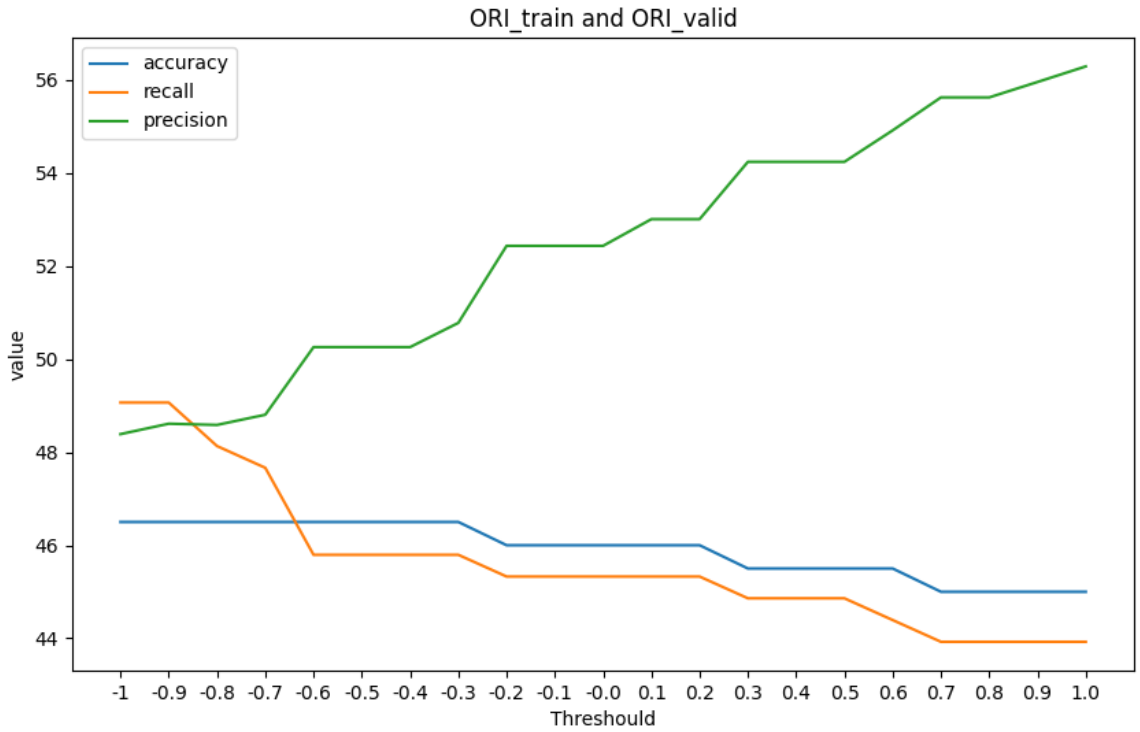
如果使用训练集的gpt产生的标签进行训练，则两个验证集的准确率均约为60%。

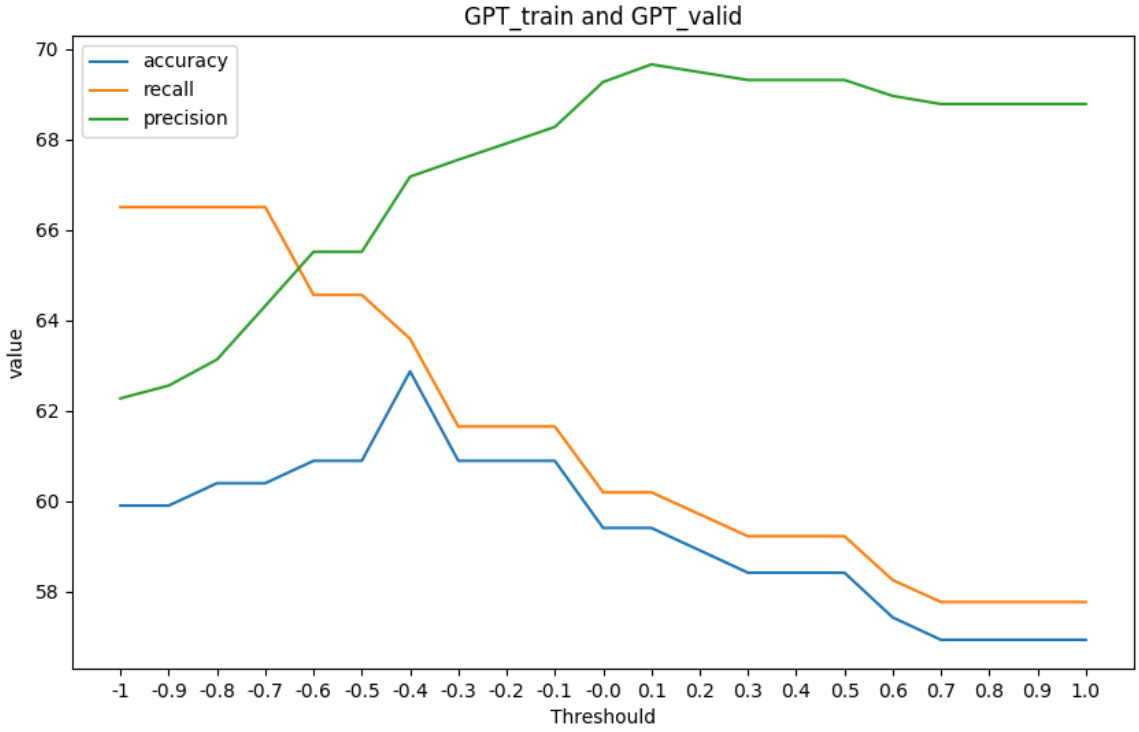
如果使用训练集的原始数据提供的标签进行训练，则GPT验证集的准确率约为40%，原始数据的验证集准确率约为45%。

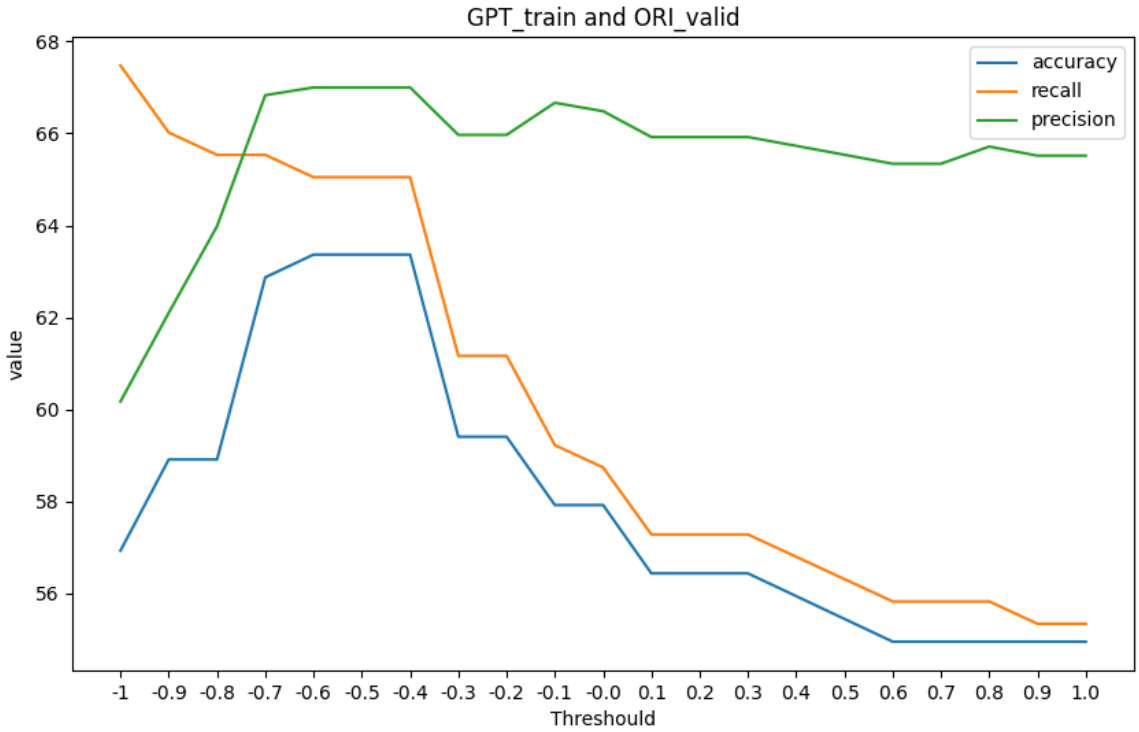
可视化表示

四组数据的验证集情况





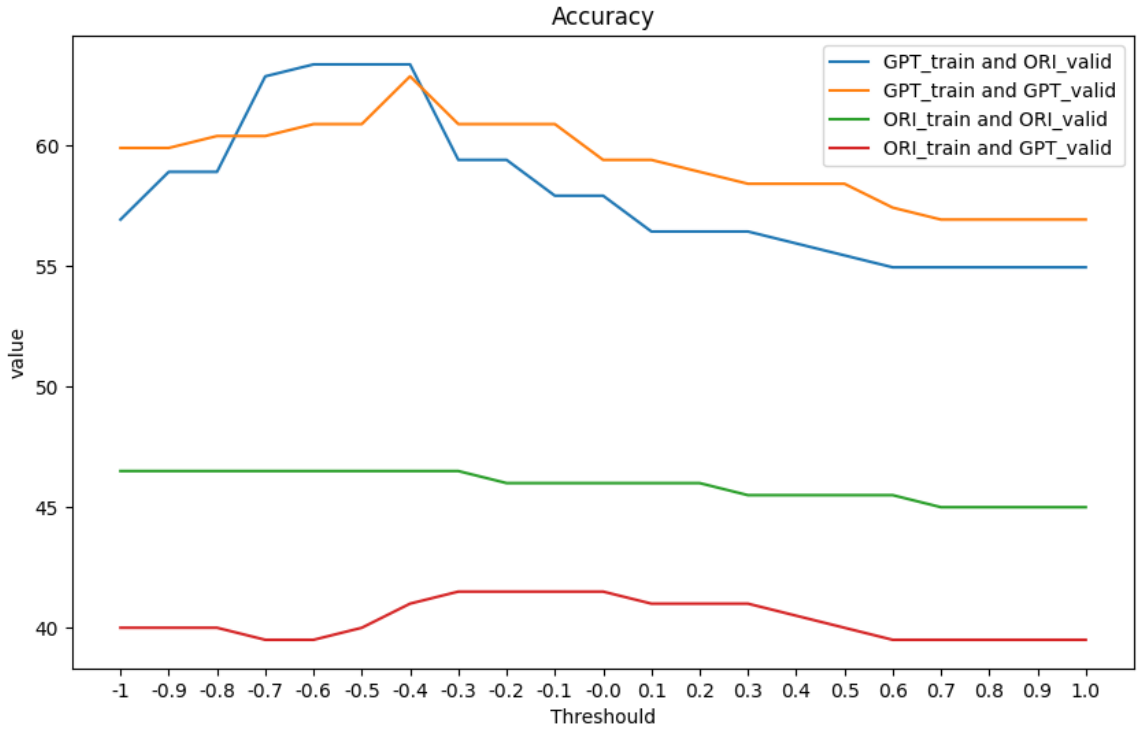




结论

经过观察，我们可以发现，对于使用原始数据作为训练集的两组数据而言，在尽可能保证准确率 (accuracy)，精确率 (precision)，召回率 (recall)的前提下，在阈值取-0.4到-0.5之间的时候，验证集的准确度差不多在40%到45%之间。而对于使用GPT数据作为训练集的话，无论验证集使用的是原始数据还是GPT数据，都有约60%的准确率。因此，采用GPT数据作为训练集是一个不错的选择。

四组数据的准确率对比



通过这个总表可以看出来，使用GPT数据作为训练集能够获得相较于使用原始数据作为训练集好很多的准确率。

错误分类样例

分类结果中"Others"出错的次数最多

	Water Parks	**Villages**	**Playgrounds**	**Others**	**Natural Landscape**	**Gardens & Parks**	**Observation Decks & Towers**	**Zoos & Animal Parks**	**Museums & Galleries**
Positive False	0	5	0	33	4	3	1	1	20
Negative False	1	0	7	26	5	11	7	0	14

其中"Others"的Negative Fasle中，许多错误都是来自于将景点误判为"Museums & Galleries"：

	Water Parks	Villages	Playgrounds	Others	Natural Landscape	Gardens & Parks	Observation Decks & Towers	Zoos & Animal Parks	Museums & Galleries
青山禅院	0	0	0	1	0	0	0	0	1
元州邨	0	1	0	1	0	0	0	0	0
纷松（马鞍山店）	0	0	0	1	0	0	0	0	1
密室梯逃	0	0	0	1	0	0	0	0	0
鲤鱼门天后庙	0	0	0	1	0	0	0	0	1
从心苑	0	0	0	1	0	0	0	0	1
滘西洲洪圣古庙	0	0	0	1	0	0	0	0	1
天后庙	0	0	0	1	0	0	0	0	1
佛堂门天后古庙	0	0	0	1	0	0	0	0	1
屏山洪圣宫	0	0	0	1	0	0	0	0	1
屏山文物径	0	0	0	1	0	0	0	0	0
妙法寺	0	0	0	1	0	0	0	0	1
维港湾会所	0	0	0	1	0	0	0	0	0
Rosie Jean's Cafe	0	0	0	1	0	0	0	0	1
Namco Wonder Park(乐富广场店)	0	0	0	1	0	0	0	0	1
玉皇宝殿	0	0	0	1	0	0	0	0	1
大角咀道/枫树街花园	0	0	0	1	0	1	0	0	0
崇谦堂	0	0	0	1	0	0	0	0	1
片蓝造	0	0	0	1	0	0	0	0	0
绿汇学苑	0	0	0	1	0	1	0	0	0
岳王古庙	0	0	0	1	0	0	0	0	1
The Hong Kong Climbing Class	0	0	0	1	0	0	1	0	0
香港单车馆公园	0	0	0	1	0	1	0	0	0
吉澳三圣宫	0	0	0	1	0	0	0	0	1
洪圣殿	0	0	0	1	0	0	0	0	1
东龙洲炮台特别地区	0	0	0	1	0	0	0	0	0

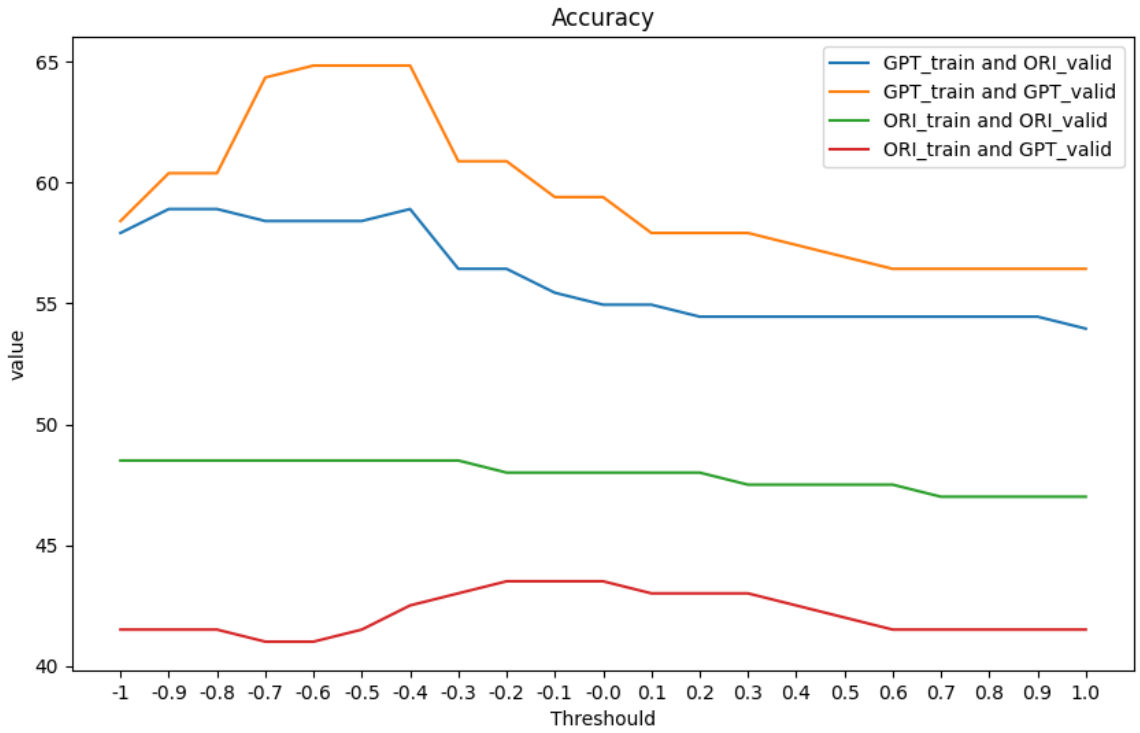
Others"的Positive Fasle中，错误分布相对比较均匀：

	Water Parks	Villages	Playgrounds	Others	Natural Landscape	Gardens & Parks	Observation Decks & Towers	Zoos & Animal Parks	Museums & Galleries
泡泡足球	0 0	0 0	1 0	0 1	0 0	0 0	0 0	0 0	0 0
欢天雪地溜冰场	0 0	0 0	1 0	0 1	0 0	0 0	0 0	0 0	0 0
深水埗运动场	0 0	0 0	1 0	0 1	0 0	0 0	0 0	0 0	0 0
香港迪士尼乐园	0 0	0 0	1 0	0 1	0 0	0 0	0 0	0 0	0 0
奥运越野障碍赛场地	0 0	0 0	1 0	0 1	0 0	0 0	0 0	0 0	0 0
锁罗盆	0 0	0 0	0 0	0 1	1 0	0 0	0 0	0 0	0 0
四柱擎天	0 0	0 0	0 0	0 1	1 0	0 0	0 0	0 0	0 0
Bohemian Garden	0 0	0 0	0 0	0 1	0 0	1 0	0 0	0 0	0 0
石澳海角郊游区	0 0	0 0	0 0	0 1	0 0	1 0	0 0	0 0	0 0
大荣有机生态园	0 0	0 0	0 0	0 1	0 0	1 0	0 0	0 0	0 0
山尾街篮球场	0 0	0 0	1 0	0 1	0 0	1 0	0 0	0 0	0 0
园圃街雀鸟花园	0 0	0 0	0 0	0 1	0 0	1 0	0 0	0 0	0 0
Kwok Shui Road Park	0 0	0 0	0 0	0 1	0 0	1 0	0 0	0 0	0 0
香港公园-海洋公园温室	0 0	0 0	0 0	0 1	0 0	0 0	0 0	0 0	1 0
上環文娛中心	0 0	0 0	0 0	0 1	0 0	0 0	0 0	0 0	1 0
KA Atelier	0 0	0 0	0 0	0 1	0 0	0 0	0 0	0 0	1 0
元朗横洲二圣宫	0 0	0 0	0 0	0 1	0 0	0 0	0 0	0 0	1 0
西湾国殇纪念坟场	0 0	0 0	0 0	0 1	0 0	0 0	0 0	0 0	1 0
赤柱北帝古庙	0 0	0 0	0 0	0 1	0 0	0 0	0 0	0 0	1 0
大埔天后宫	0 0	0 0	0 0	0 1	0 0	0 0	0 0	0 0	1 0
松岭邓公祠	0 0	0 0	0 0	0 1	0 0	0 0	0 0	0 0	1 0
Gallery One	0 0	0 0	0 0	0 1	0 0	0 0	0 0	0 0	1 0
甄子杰夫人纪念堂	0 0	0 0	0 0	0 1	0 0	0 0	0 0	0 0	1 0
愈乔二公祠	0 0	0 0	0 0	0 1	0 0	0 0	0 0	0 0	1 0
黄大仙祠初一祈福活动	0 0	0 0	0 0	0 1	0 0	0 0	0 0	0 0	1 0
太安楼	0 0	0 0	0 0	0 1	0 0	0 0	1 0	0 0	0 0
孖岗山观景台	0 0	0 0	0 0	0 1	0 0	0 0	1 1	0 0	0 0
長洲觀音灣觀海亭	0 0	0 0	0 0	0 1	0 0	0 0	1 0	0 0	0 0
虎山观景台	0 0	0 0	0 0	0 1	0 0	0 0	1 0	0 0	0 0
飞鹅山观景台	0 0	0 0	0 0	0 1	0 0	0 0	1 0	0 0	0 0
石壁观景台	0 0	0 0	0 0	0 1	0 0	0 0	1 0	0 0	0 0
尤德观鸟园	0 0	0 0	0 0	0 1	0 0	0 0	0 0	1 0	0 0
瘋狂過山車	1 0	0 0	0 0	0 1	0 0	0 0	0 0	0 0	0 0

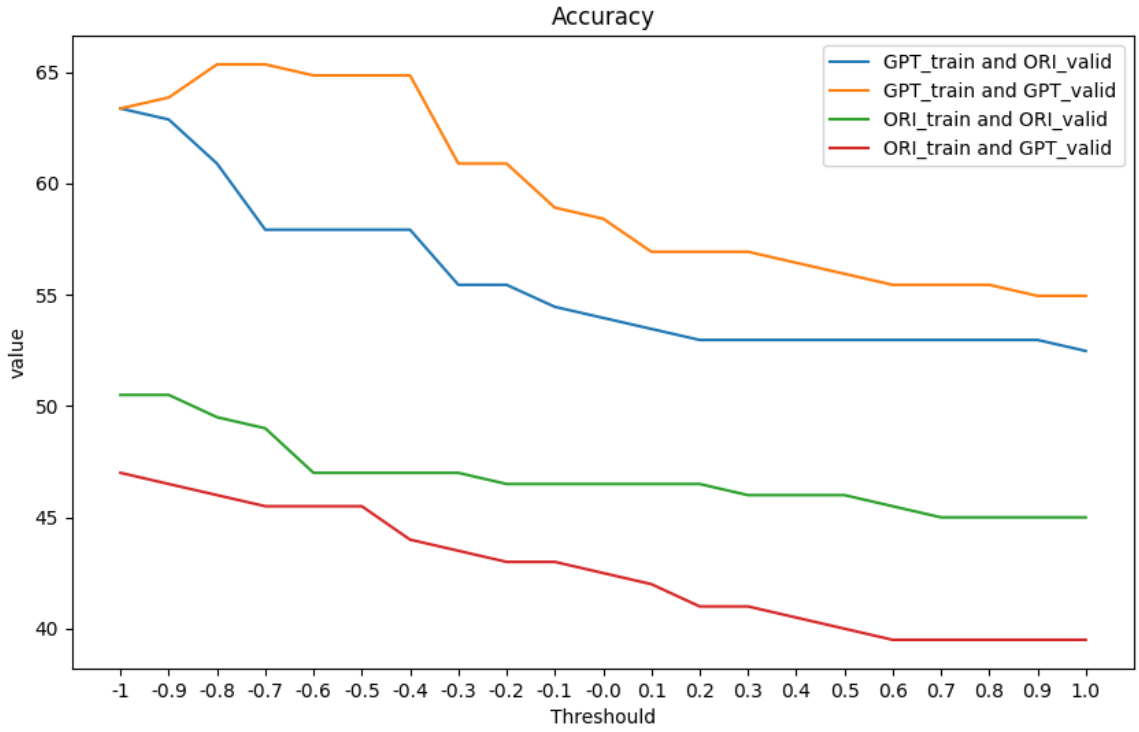
其它数据分析补充

如果采用更宽松的判断标准：

1.只有预测的结果中有不应该出现的结果时才判为失败，也就是允许少判断标签，例如正确标签是1和2，则预测为1或这预测为2都不算错误，只有出现了1和2之外的预测才算错误，同时如果标签都是0则也算预测错误。



2.允许预测结果出现最多一个不一样的结果，但是不允许标签的缺少。



详细数据文件：

ORI_train_gpt_vaild_combination.jsonORI_train_ori_vaild_combination.json