【概述】

Qwen1.5大模型微调、基于PEFT框架LoRA微调，在数据集HC3-Chinese上实现文本分类。

运行环境：Kaggle - Notebook

【数据处理】

1.【数据下载】

```python
import modelscope
from modelscope.msdatasets import MsDataset
#【下载数据集】
HC3=MsDataset.load('simpleai/HC3-Chinese',subset_name='baike',split='train') #调用HC3数据集
dataset=HC3.to_hf_dataset() #将MsDataset转换成huggingface dataset格式，方便后续处理
print("【数据集下载完成】")
print(dataset)
print(dataset[0])
```

输出信息：

```
Dataset({
    features: ['id', 'question', 'human_answers', 'chatgpt_answers'],
    num_rows: 4617
})

{
    'id': '0',
    'question': '我有一个计算机相关的问题...',
    'human_answers': ['硬盘安装就是...'],
    'chatgpt_answers': ['硬盘安装是指...']
}
```

2.【格式调整】

将数据调整成形如`{label: 0/1, text: '...'}`的格式，在 9234 组数据中随机选 5000 个，按照 8：1：1 的比例划分训练集、验证集、测试集。

```python
from datasets import Dataset
#【调整数据集格式】
def data_init(dataset):
    ds=[]
    cnt=dataset.num_rows
    for i in range(cnt):
        example=dataset[i]
        ds.append({"label":0,"text":example["human_answers"][0]})
        ds.append({"label":1,"text":example["chatgpt_answers"][0]})
    return Dataset.from_list(ds)

dataset=data_init(dataset) # 调整数据集内容
print(dataset)
dataset=dataset.shuffle(seed=233).select(range(5000)) #随机选一部分

#数据集划分 train:val:test=8:1:1
data_=dataset.train_test_split(train_size=0.8,seed=233) #数据集划分
data_train=data_["train"]
data__=data_["test"].train_test_split(train_size=0.5,seed=233)
data_val=data__["train"]
data_test=data__["test"]

print("【data_train】",data_train)
print("【data_val】",data_val)
print("【data_test】",data_test)
```

输出信息：

```
Dataset({
    features: ['label', 'text'],
    num_rows: 9234
})
【data_train】 Dataset({
```

```
    features: ['label', 'text'],
    num_rows: 4000
})
【data_val】 Dataset({
    features: ['label', 'text'],
    num_rows: 500
})
【data_test】 Dataset({
    features: ['label', 'text'],
    num_rows: 500
})
```

【模型】

1.【分词器】

文本信息在输入模型前，需要先用tokenizer分词。使用Dataset.map()函数快速处理。

```
from transformers import AutoTokenizer,AutoModelForSequenceClassification,TrainingArguments,Trainer,DataCollatorWithPadding

#【加载分词器】
tokenizer = AutoTokenizer.from_pretrained("Qwen/Qwen1.5-0.5B")
tokenizer.pad_token_id = tokenizer.eos_token_id #Qwen特性，需要指定一下pad_token_id

def tokenize_function(examples):
    return tokenizer(examples["text"],padding="max_length",truncation=True,max_length=512)

token_train=data_train.map(tokenize_function, batched=True)
token_val=data_val.map(tokenize_function, batched=True)

train_dataset = token_train
eval_dataset = token_val
```

2.【加载模型】

用`AutoModelForSequenceClassification`载入模型进行文本分类任务。`num_labels`为要分类的标签数量。

`from_pretrained()` 支持的模型在这里可以找到。

> 报错 `KeyError: 'qwen2'` 应该是 `transformers` 版本太旧。

```
#【加载模型】
id2label = {0: "human", 1: "chatgpt"}
label2id = {"human": 0, "chatgpt": 1}
#使用Qwen1.5模型
model = AutoModelForSequenceClassification.from_pretrained("Qwen/Qwen1.5-0.5B",num_labels=2,id2label=id2label,label2id=label2id)
model.config.pad_token_id=model.config.eos_token_id #这里也要指定一下pad_token_id，不然训练时会报错 "ValueError: Cannot handle batch sizes > 1 if no padding token is defined."
print("【model】\n",model)
print("【model.config】\n",model.config)
```

输出信息可以看到模型结构，以及 `pad_token_id`（如果没有指定的话可以看到 `config` 里没有这个变量）

```
【model】
 Qwen2ForSequenceClassification(
  (model): Qwen2Model(
    (embed_tokens): Embedding(151936, 1024)
    (layers): ModuleList(
      (0-23): 24 x Qwen2DecoderLayer(
        (self_attn): Qwen2SdpaAttention(
          (q_proj): Linear(in_features=1024, out_features=1024, bias=True)
          (k_proj): Linear(in_features=1024, out_features=1024, bias=True)
          (v_proj): Linear(in_features=1024, out_features=1024, bias=True)
          (o_proj): Linear(in_features=1024, out_features=1024, bias=False)
          (rotary_emb): Qwen2RotaryEmbedding()
        )
        (mlp): Qwen2MLP(
```

```
          (gate_proj): Linear(in_features=1024, out_features=2816, bias=False)
          (up_proj): Linear(in_features=1024, out_features=2816, bias=False)
          (down_proj): Linear(in_features=2816, out_features=1024, bias=False)
          (act_fn): SiLU()
        )
        (input_layernorm): Qwen2RMSNorm()
        (post_attention_layernorm): Qwen2RMSNorm()
      )
    )
    (norm): Qwen2RMSNorm()
  )
  (score): Linear(in_features=1024, out_features=2, bias=False)
)
【model.config】
 Qwen2Config {
  "_name_or_path": "Qwen/Qwen1.5-0.5B",
  "architectures": [
    "Qwen2ForCausalLM"
  ],
  "attention_dropout": 0.0,
  "bos_token_id": 151643,
  "eos_token_id": 151643,
  "hidden_act": "silu",
  "hidden_size": 1024,
  "id2label": {
    "0": "human",
    "1": "chatgpt"
  },
  "initializer_range": 0.02,
  "intermediate_size": 2816,
  "label2id": {
    "chatgpt": 1,
    "human": 0
  },
  "max_position_embeddings": 32768,
  "max_window_layers": 21,
  "model_type": "qwen2",
  "num_attention_heads": 16,
  "num_hidden_layers": 24,
```

```
  "num_key_value_heads": 16,
  "pad_token_id": 151643,
  "rms_norm_eps": 1e-06,
  "rope_theta": 1000000.0,
  "sliding_window": 32768,
  "tie_word_embeddings": true,
  "torch_dtype": "bfloat16",
  "transformers_version": "4.41.2",
  "use_cache": true,
  "use_sliding_window": false,
  "vocab_size": 151936
}
```

【训练】

1.【训练参数】

```
#【训练参数】
from datasets import load_metric
import numpy as np

training_args = TrainingArguments(
    output_dir="pt_save_pretrained",
    evaluation_strategy="epoch", #每跑完一个epoch输出一下测试信息
    num_train_epochs=2,
    per_device_train_batch_size=4, # 一共要跑 len(dataset)/batch_size * epoch 个step
                                  # [模型=Qwen1.5-0.5B, batch_size=4]：完全微调显存13.3GB，LoRA微调显存8.7GB
    save_strategy="no",  #关闭自动保存模型（Kaggle上磁盘空间不太够）
)

metric=load_metric('accuracy') #评估指标

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return metric.compute(predictions=predictions, references=labels)

def get_trainer(model):
```

```
    return  Trainer(
        model=model,
        args=training_args,
        tokenizer=tokenizer,
        train_dataset=train_dataset,
        eval_dataset=eval_dataset,
        compute_metrics=compute_metrics,
        data_collator=DataCollatorWithPadding(tokenizer=tokenizer, padding=True,return_tensors="pt"),  #给数据添加padding弄成batch
    )
```

2.【完全微调】

直接开始训练:

```
#【完全微调】
print("【开始训练】")
trainer=get_trainer(model)
trainer.train()

#tokenizer.save_pretrained("./full_model_tokenizer")
#model.save_pretrained("./full_model")

#Kaggle注意:
每次训练之后restart以释放显存!
factory也reset一下，不然磁盘空间会爆!
```

训练效果:

[2000/2000 30:07, Epoch 2/2]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.304600	0.217201	0.956000
2	0.099500	0.148824	0.962000

3.【LoRA微调】

添加 LoRA 参数，调用peft框架:

```
#【PEFT-LoRA微调】
from peft import LoraConfig, get_peft_model
```

```python

peft_config = LoraConfig(
    task_type="SEQ_CLS", #任务类型：分类
    target_modules=["q_proh","k_proj","v_proj","o_proj"], # 这个不同的模型需要设置不同的参数，主要看模型中的attention层
    inference_mode=False, # 关闭推理模式 (即开启训练模式)
    r=8, # Lora 秩
    lora_alpha=16, # Lora alaph，具体作用参见 Lora 原理
    lora_dropout=0.1 # Dropout 比例
)

peft_model = get_peft_model(model, peft_config) # 加载lora参数peft框架

print('PEFT参数量：')
peft_model.print_trainable_parameters()

print("【开始训练】")
peft_trainer=get_trainer(peft_model)
peft_trainer.train()

tokenizer.save_pretrained("./peft_model_tokenizer")
peft_model.save_pretrained("./peft_model")
```

从输出结果可以看到，LoRA 微调所要训练的参数只占 25.4%，显著降低显存占用和训练时间：

```
PEFT参数量：
trainable params: 1,181,696 || all params: 465,171,456 || trainable%: 0.2540
```

训练效果：

[2000/2000 23:08, Epoch 2/2]

Epoch	Training Loss	Validation Loss	Accuracy
1	0.102500	0.113912	0.986000
2	0.052600	0.107569	0.986000

【测试】

1.【代码】

```python
import torch
```

```
from transformers import DataCollatorWithPadding,AutoTokenizer,AutoModelForSequenceClassification

def classify(example,show): #对example进行预测
    text=example["text"]
    label=example["label"]
    inputs = tokenizer(text, truncation=True, padding=True, return_tensors="pt").to('cuda')
    with torch.no_grad():
        output = inference_model(**inputs)
        pred = output.logits.argmax(dim=-1).item()
    if show:
        print("【预测{}!】Label: {}, Pred_Label: {}\nText: {}".format("正确" if label==pred else "错误",id2label[label],id2label[pred],text))
    else:
        return pred,label

#inference_model=model.to('cuda')
tokenizer = AutoTokenizer.from_pretrained("./peft_model_tokenizer")
inference_model = AutoModelForSequenceClassification.from_pretrained("./peft_model").to('cuda') #读取训练好的模型
print("【model】\n",inference_model)
print("【model.config】\n",inference_model.config)
print("【model.config.pad_token_id】",inference_model.config.pad_token_id)
data_collator=DataCollatorWithPadding(tokenizer=tokenizer, padding=True,return_tensors="pt")


id2label = {0: "human", 1: "chatgpt"}
label2id = {"human": 0, "chatgpt": 1}

classify(data_test[0],1) #随便测试一个数据
```

输出：

```
【预测正确!】Label: human, Pred_Label: human
Text: 硬盘接口是硬盘与主机系统间的连接部件，作用是在硬盘缓存和主机内存之间传输数据。不同的硬盘接口决定着硬盘与计算机之间的连接速度，在整个系统中，硬盘接口的优劣直接影响着程序运行快慢和系统性能好坏。
```

在测试集上评估性能，二分类使用`accuracy`指标：

```
from datasets import load_metric
from tqdm import tqdm
metric=load_metric('accuracy')

print("【测试集】",data_test)
inference_model.eval()
for i,example in enumerate(tqdm(data_test)):
    pred, label = classify(example,0)
    metric.add(predictions=pred, references=label)
print(metric.compute())
```

2.【结果展示】

使用 `Qwen1.5-1.8B-Chat` 、给予 `prompt` 为 `{system="你擅长文本分类，能判断一段文字描述来自human还是chatgpt。你只需要回答一个单词，回答human或者chatgpt。", user=这里有一段关于xxx的文字描述，请判断它来自human还是chatgpt：\n xxx"}`，输出基本全是 `human`。

```
测试集大小： 30*2
Test NO.0: response=[人类,human] acc=(1/2)
Test NO.1: response=[人类,human] acc=(2/4)
Test NO.2: response=[human,human] acc=(3/6)
Test NO.3: response=[chatgpt,human] acc=(3/8)
Test NO.4: response=[human,human] acc=(4/10)
Test NO.5: response=[human,人类] acc=(5/12)
Test NO.6: response=[human,human] acc=(6/14)
Test NO.7: response=[human,human] acc=(7/16)
Test NO.8: response=[human,human] acc=(8/18)
Test NO.9: response=[human,human] acc=(9/20)
Test NO.10: response=[human,人类] acc=(10/22)
Test NO.11: response=[human,human] acc=(11/24)
Test NO.12: response=[chatgpt,human] acc=(11/26)
Test NO.13: response=[human,human] acc=(12/28)
Test NO.14: response=[human,human] acc=(13/30)
Test NO.15: response=[human,人类] acc=(14/32)
```

```
Test NO.16: response=[human,human] acc=(15/34)
Test NO.17: response=[人类,人类] acc=(16/36)
Test NO.18: response=[human,human] acc=(17/38)
Test NO.19: response=[human,human] acc=(18/40)
Test NO.20: response=[chatgpt,人类] acc=(18/42)
Test NO.21: response=[human,human] acc=(19/44)
Test NO.22: response=[chatgpt,human] acc=(19/46)
Test NO.23: response=[human,human] acc=(20/48)
Test NO.24: response=[human,human] acc=(21/50)
Test NO.25: response=[human,人类] acc=(22/52)
Test NO.26: response=[human,human] acc=(23/54)
Test NO.27: response=[human,human] acc=(24/56)
Test NO.28: response=[human,human] acc=(25/58)
Test NO.29: response=[人类,人类] acc=(26/60)

Test准确率：acc=0.43333333333333335
```

`Qwen1.5-0.5B` 完全微调：

```
【测试集】 Dataset({
    features: ['label', 'text'],
    num_rows: 500
})

{'accuracy': 0.944}
```

`Qwen1.5-0.5B` Lora微调：

```
【测试集】 Dataset({
    features: ['label', 'text'],
    num_rows: 500
})

{'accuracy': 0.982}

```