博学谷
www.boxuegu.com
黑马程序员线上品牌

基于BERT+P-Tuning方式数据预处理

一样的教育，不一样的品质

目录

Contents

01

查看项目数据集

数据集简介

数据存放位置：/Users/**/PycharmProjects/llm/prompt_tasks/P-Tuning/data

data文件夹里面包含3个txt文档，分别为：train.txt、dev.txt、verbalizer.txt

train.txt

train.txt为训练数据集，其部分数据展示如下

```
水果	脆脆的，甜味可以，可能时间有点长了，水分不是很足。
平板	华为机器肯定不错，但第一次碰上京东最糟糕的服务，以后不想到京东购物了。
书籍	为什么不认真的检查一下， 发这么一本脏脏的书给顾客呢!
衣服	手感不错，用料也很好，不知道水洗后怎样，相信大品牌，质量过关，五星好评！！！
水果	苹果有点小，不过好吃，还有几个烂的。估计是故意的放的。差评。
衣服	掉色掉的厉害，洗一次就花了
```

train.txt一共包含63条样本数据，每一行用`\t`分开，前半部分为标签(label)，后半部分为原始输入 (用户评论)。

如果想使用自定义数据训练，只需要仿照上述示例数据构建数据集即可。

dev.txt

dev.txt为验证数据集，其部分数据展示如下

书籍	"一点都不好笑,很失望,内容也不是很实用"
衣服	完全是一条旧裤子。
手机	相机质量不错，如果阳光充足，可以和数码相机媲美．界面比较人性化，容易使用．软件安装简便
书籍	明明说有货，结果送货又没有了。并且也不告诉我，怎么评啊
洗浴	非常不满意，晚上洗的头发，第二天头痒痒的不行了，还都是头皮屑。
水果	这个苹果感觉是长熟的苹果，没有打蜡，不错，又甜又脆

dev.txt一共包含417条样本数据，每一行用\t分开，前半部分为标签(label)，后半部分为原始输入 (用户评论)。

如果想使用自定义数据训练，只需要仿照上述示例数据构建数据集即可。

verbalizer.txt

- verbalizer.txt 主要用于定义「真实标签」到「标签预测词」之间的映射。在有些情况下，将「真实标签」作为 [MASK] 去预测可能不具备很好的语义通顺性，因此，我们会对「真实标签」做一定的映射。
- 例如

```
"中国爆冷2–1战胜韩国"是一则[MASK][MASK]新闻。        体育
```

- 这句话中的标签为「体育」，但如果我们将标签设置为「足球」会更容易预测。
- 因此，我们可以对「体育」这个 label 构建许多个子标签，在推理时，只要预测到子标签最终推理出真实标签即可，如下

```
体育 -> 足球,篮球,网球,棒球,乒乓,体育
```

verbalizer.txt

- 项目中标签词映射数据展示如下：

电脑	电脑
水果	水果
平板	平板
衣服	衣服
酒店	酒店
洗浴	洗浴
书籍	书籍
蒙牛	蒙牛
手机	手机
电器	电器

verbalizer.txt 一共包含10个类别，上述数据中，我们使用了1对1的verbalizer, 如果想定义一对多的映射，只需要在后面用","分割即可， eg:

水果　　苹果,香蕉,橘子

若想使用自定义数据训练，只需要仿照示例数据构建数据集

02

编写Config类项目文件配置代码

项目文件简介

代码路径

/Users/**/PycharmProjects/llm/prompt_tasks/P-Tuning/ptune_config.py

config文件目的

配置项目常用变量，一般这些变量属于不经常改变的，比如：训练文件路径、模型训练次数、模型超参数等等

具体代码实现

```
# coding:utf-8
import torch
class ProjectConfig(object):
    def __init__(self):
        self.device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
        self.pre_model = './bert-base-chinese'
        self.train_path = './data/train.txt'
        self.dev_path = './data/dev.txt'
        self.verbalizer = './data/verbalizer.txt'
        self.max_seq_len = 512
        self.batch_size = 8
        self.learning_rate = 5e-5
        self.weight_decay = 0
        self.warmup_ratio = 0.06
        self.p_embedding_num = 6
        self.max_label_len = 2
        self.epochs = 50
        self.logging_steps = 10
        self.valid_steps = 20
        self.save_dir = './P-Tuning/checkpoints'

if __name__ == '__main__':
    pc = ProjectConfig()
    print(pc.verbalizer)
```

03

编写数据处理相关代码

代码简介

- 代码路径：
  /Users/***/PycharmProjects/llm/prompt_tasks/P-Tuning/data_handle.

- data_handle文件夹中一共包含两个py脚本：
  data_preprocess.py、data_loader.py

data_preprocess.py

目的：将样本数据转换为模型接受的输入数据

导入必备的工具包

```
# 导入必备工具包
import torch
import numpy as np
from rich import print
from datasets import load_dataset
from transformers import AutoTokenizer
import sys
sys.path.append('..')
from ptune_config import *
from functools import partial
```

data_preprocess.py

定义数据转换方法convert_example()

目的：将模板与原始输入文本进行拼接，构造模型接受的输入数据

data_preprocess.py

打印结果展示

```
{
 'input_ids': array([[   1,    2,    3, ..., 1912, 6225,
102],
    [   1,    2,    3, ..., 3300, 5741,  102],
    [   1,    2,    3, ..., 6574, 7030,    0],
    ...,
    [   1,    2,    3, ..., 8024, 2571,    0],
    [   1,    2,    3, ..., 3221, 3175,  102],
    [   1,    2,    3, ..., 5277, 3688,  102]]),
 'attention_mask': array([[1, 1, 1, ..., 1, 1, 1],
    [1, 1, 1, ..., 1, 1, 1],
    [1, 1, 1, ..., 0, 0, 0],
    ...,
    [1, 1, 1, ..., 0, 0, 0],
    [1, 1, 1, ..., 1, 1, 1],
    [1, 1, 1, ..., 1, 1, 1]]),
 'mask_positions': array([[7, 8],
    [7, 8],
    [7, 8],
    ...,
    [7, 8],
    [7, 8],
    [7, 8]]),
 'mask_labels': array([[4510, 5554],
    [3717, 3362],
    [2398, 3352],
    ...,
    [3819, 3861],
    [6983, 2421],
    [3819, 3861]]),
 'token_type_ids': array([[0, 0, 0, ..., 0, 0, 0],
    [0, 0, 0, ..., 0, 0, 0],
    [0, 0, 0, ..., 0, 0, 0],
    ...,
    [0, 0, 0, ..., 0, 0, 0],
    [0, 0, 0, ..., 0, 0, 0],
    [0, 0, 0, ..., 0, 0, 0]])
}
```

data_loader.py

目的：定义数据加载器

导入必备的工具包

```
# coding:utf-8
from torch.utils.data import DataLoader
from transformers import default_data_collator, AutoTokenizer
from data_handle.data_preprocess import *
from ptune_config import *

pc = ProjectConfig() # 实例化项目配置文件

tokenizer = AutoTokenizer.from_pretrained(pc.pre_model)
```

data_loader.py

定义获取数据加载器的方法get_data()

```
def get_data():
    dataset = load_dataset('text', data_files={'train': pc.train_path,
                                               'dev': pc.dev_path})
    new_func = partial(convert_example,
                       tokenizer=tokenizer,
                       max_seq_len=pc.max_seq_len,
                       max_label_len=pc.max_label_len,
                       p_embedding_num=pc.p_embedding_num)

    dataset = dataset.map(new_func, batched=True)
    train_dataset = dataset["train"]
    dev_dataset = dataset["dev"]
    train_dataloader = DataLoader(train_dataset,
                                  shuffle=True,

collate_fn=default_data_collator,
                                  batch_size=pc.batch_size)
    dev_dataloader = DataLoader(dev_dataset,
                                collate_fn=default_data_collator,
                                batch_size=pc.batch_size)
    return train_dataloader, dev_dataloader


if __name__ == '__main__':
    train_dataloader, dev_dataloader = get_data()
    print(len(train_dataloader))
    print(len(dev_dataloader))
    for i, value in enumerate(train_dataloader):
        print(i)
        print(value)
        print(value['input_ids'].dtype)
        break
```

data_loader.py

打印结果展示

```
{
 'input_ids': tensor([[1, 2, 3,  ..., 0, 0, 0],
        [1, 2, 3,  ..., 0, 0, 0],
        [1, 2, 3,  ..., 0, 0, 0],
        ...,
        [1, 2, 3,  ..., 0, 0, 0],
        [1, 2, 3,  ..., 0, 0, 0],
        [1, 2, 3,  ..., 0, 0, 0]]),
 'attention_mask': tensor([[1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        ...,
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0],
        [1, 1, 1,  ..., 0, 0, 0]]),
        'mask_positions': tensor([[7, 8],
        [7, 8],
        [7, 8],
        [7, 8],
        [7, 8],
        [7, 8],
        [7, 8]]),
 'mask_labels': tensor([[6132, 3302],
        [2398, 3352],
        [6132, 3302],
        [6983, 2421],
        [3717, 3362],
        [6983, 2421],
        [3819, 3861],
        [6983, 2421]]),
 'token_type_ids': tensor([[0, 0, 0,  ..., 0, 0, 0],
        [0, 0, 0,  ..., 0, 0, 0],
        [0, 0, 0,  ..., 0, 0, 0],
        ...,
        [0, 0, 0,  ..., 0, 0, 0],
        [0, 0, 0,  ..., 0, 0, 0],
        [0, 0, 0,  ..., 0, 0, 0]])
}
torch.int64
```

总结

sum up

- 介绍了基于BERT+P-Tuning方式实现文本分类任务时数据处理步骤
- 通过代码实现：提示模板数据格式的转换，数据加载器的编码等

博学谷
www.boxuegu.com
黑马程序员线上品牌

Thanks!

扫码关注博学谷微信公众号