基于BERT+PET方式数据预处理介绍

学习目标

- 了解本项目数据类型和表现格式
- 掌握数据处理的工具函数代码实现

BERT+PET方式数据预处理

- 本项目中对数据部分的预处理步骤如下:
    1. 查看项目数据集
    2. 编写Config类项目文件配置代码
    3. 编写数据处理相关代码

1 查看项目数据集

- 数据存放位置：/Users/***/PycharmProjects/llm/prompt_tasks/PET/data
- data文件夹里面包含4个txt文档，分别为：train.txt、dev.txt、prompt.txt、verbalizer.txt

---

1.1 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。

1.2 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590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。

1.3 prompt.txt

- prompt.txt为人工设定提示模版，其数据展示如下：

```
这是一条{MASK}评论：{textA}。
```

> 其中，用大括号括起来的部分为「自定义参数」，可以自定义设置大括号内的值。

> 示例中 {MASK} 代表 [MASK] token 的位置，{textA} 代表评论数据的位置。
>
> 你可以改为自己想要的模板，例如想新增一个 {textB} 参数：
>
> ```
> {textA}和{textB}是{MASK}同的意思。
> ```

1.4 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
```

- 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

2 编写Config类项目文件配置代码

- 代码路径：/Users/***/PycharmProjects/llm/prompt_tasks/PET/pet_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：

```python
# coding:utf-8
import torch
import sys
print(sys.path)

class ProjectConfig(object):
    def __init__(self):
        # 是否使用GPU
        self.device = 'cuda:0' if torch.cuda.is_available() else 'cpu'
        # 预训练模型bert路径
        self.pre_model = '/home/prompt_project/bert-base-chinese'
        self.train_path = '/home/prompt_project/PET/data/train.txt'
        self.dev_path = '/home/prompt_project/PET/data/dev.txt'
        self.prompt_file = '/home/prompt_project/PET/data/prompt.txt'
        self.verbalizer = '/home/prompt_project/PET/data/verbalizer.txt'
        self.max_seq_len = 512
        self.batch_size = 8
        self.learning_rate = 5e-5
        # 权重衰减参数（正则化，抑制模型过拟合）
        self.weight_decay = 0
        # 预热学习率(用来定义预热的步数)
        self.warmup_ratio = 0.06
```

```
        self.max_label_len = 2
        self.epochs = 50
        self.logging_steps = 10
        self.valid_steps = 20
        self.save_dir = '/home/prompt_project/PET/checkpoints'


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

3 编写数据处理相关代码

- 代码路径：/Users/***/PycharmProjects/llm/prompt_tasks/PET/data_handle.
- data_handle文件夹中一共包含三个py脚本：template.py、data_preprocess.py、data_loader.py

3.1 template.py

- 目的：构建固定模版类，text2id的转换
- 导入必备工具包

```
# -*- coding:utf-8 -*-
from rich import print # 终端层次显示
from transformers import AutoTokenizer
import numpy as np
from pet_config import *
```

- 定义HardTemplate类

```
class HardTemplate(object):
    """
    硬模板，人工定义句子和[MASK]之间的位置关系。
    """

    def __init__(self, prompt: str):
        """
        Args:
            prompt (str): prompt格式定义字符串，e.g. -> "这是一条{MASK}评论:{textA}。"
        """
        self.prompt = prompt
        self.inputs_list = []                       # 根据文字prompt拆分为各part的列表
        self.custom_tokens = set(['MASK'])          # 从prompt中解析出的自定义token集合
        self.prompt_analysis()                      # 解析prompt模板
```

```
15
16      def prompt_analysis(self):
17          """
18          将prompt文字模板拆解为可映射的数据结构。
19
20          Examples:
21           prompt -> "这是一条{MASK}评论：{textA}。"
22           inputs_list -> ['这', '是', '一', '条', 'MASK', '评', '论', '：',
    'textA', '。']
23           custom_tokens -> {'textA', 'MASK'}
24          """
25          idx = 0
26          while idx < len(self.prompt):
27              str_part = ''
28              if self.prompt[idx] not in ['{', '}']:
29                  self.inputs_list.append(self.prompt[idx])
30              if self.prompt[idx] == '{':                    # 进入自定义字段
31                  idx += 1
32                  while self.prompt[idx] != '}':
33                      str_part += self.prompt[idx]              # 拼接该自定义
    字段的值
34                      idx += 1
35              elif self.prompt[idx] == '}':
36                  raise ValueError("Unmatched bracket '}', check your
    prompt.")
37              if str_part:
38                  self.inputs_list.append(str_part)
39                  # 将所有自定义字段存储，后续会检测输入信息是否完整
40                  self.custom_tokens.add(str_part)
41              idx += 1
42
43      def __call__(self,
44                   inputs_dict: dict,
45                   tokenizer,
46                   mask_length,
47                   max_seq_len=512):
48          """
49          输入一个样本，转换为符合模板的格式。
50
51          Args:
52              inputs_dict (dict): prompt中的参数字典, e.g. -> {
53                                                                  "textA": "这
    个手机也太卡了",
54                                                                  "MASK": "
    [MASK]"
55                                                              }
56              tokenizer: 用于encoding文本
57              mask_length (int): MASK token 的长度
58
```

```
        Returns:
            dict -> {
                'text': '[CLS]这是一条[MASK]评论：这个手机也太卡了。[SEP]',
                'input_ids': [1, 47, 10, 7, 304, 3, 480, 279, 74, 47,
                              27, 247, 98, 105, 512, 777, 15, 12043, 2],
                'token_type_ids': [0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                   0, 0, 0, 0, 0, 0, 0, 0, 0],
                'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1,
                                   1, 1, 1, 1, 1, 1, 1, 1, 1, 1],
                'mask_position': [0, 0, 0, 0, 0, 1, 0, 0, 0, 0,
                                  0, 0, 0, 0, 0, 0, 0, 0, 0]
            }
        """
        # 定义输出格式
        outputs = {
            'text': '',
            'input_ids': [],
            'token_type_ids': [],
            'attention_mask': [],
            'mask_position': []
        }

        str_formated = ''
        for value in self.inputs_list:
            if value in self.custom_tokens:
                if value == 'MASK':
                    str_formated += inputs_dict[value] * mask_length
                else:
                    str_formated += inputs_dict[value]
            else:
                str_formated += value
        # print(f'str_formated-->{str_formated}')
        encoded = tokenizer(text=str_formated,
                            truncation=True,
                            max_length=max_seq_len,
                            padding='max_length')
        # print(f'encoded--->{encoded}')
        outputs['input_ids'] = encoded['input_ids']
        outputs['token_type_ids'] = encoded['token_type_ids']
        outputs['attention_mask'] = encoded['attention_mask']
        token_list = tokenizer.convert_ids_to_tokens(encoded['input_ids'])
        outputs['text'] = ''.join(token_list)
        mask_token_id = tokenizer.convert_tokens_to_ids(['[MASK]'])[0]
        condition = np.array(outputs['input_ids']) == mask_token_id
        mask_position = np.where(condition)[0].tolist()
        outputs['mask_position'] = mask_position
        return outputs

```

```
107
108 if __name__ == '__main__':
109     pc = ProjectConfig()
110     tokenizer = AutoTokenizer.from_pretrained(pc.pre_model)
111     hard_template = HardTemplate(prompt='这是一条{MASK}评论：{textA}')
112     print(hard_template.inputs_list)
113     print(hard_template.custom_tokens)
114     tep = hard_template(
115                 inputs_dict={'textA': '包装不错，苹果挺甜的，个头也大。',
    'MASK': '[MASK]'},
116                 tokenizer=tokenizer,
117                 max_seq_len=30,
118                 mask_length=2)
119     print(tep)
120
121     print(tokenizer.convert_ids_to_tokens([3819, 3352]))
122     print(tokenizer.convert_tokens_to_ids(['水', '果']))
```

3.2 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

```
1 from template import *
2 from rich import print
3 from datasets import load_dataset
4 # partial：把一个函数的某些参数给固定住（也就是设置默认值），返回一个新的函数，调用这个新
  函数会更简单
5 from functools import partial
6 from pet_config import *
```

- 定义数据转换方法convert_example()

```
1  def convert_example(
2          examples: dict,
3          tokenizer,
4          max_seq_len: int,
5          max_label_len: int,
6          hard_template: HardTemplate,
7          train_mode=True,
8          return_tensor=False) -> dict:
9      """
10     将样本数据转换为模型接收的输入数据。
11
12     Args:
13         examples (dict): 训练数据样本, e.g. -> {
```

```
                                                "text": [
                                                            '手机 这个手机也太卡了。',
                                                            '体育 世界杯为何迟迟不见宣传',
                                                            ...
                                                ]
                                            }
        max_seq_len (int): 句子的最大长度，若没有达到最大长度，则padding为最大长度
        max_label_len (int): 最大label长度，若没有达到最大长度，则padding为最大长度
        hard_template (HardTemplate): 模板类。
        train_mode (bool): 训练阶段 or 推理阶段。
        return_tensor (bool): 是否返回tensor类型，如不是，则返回numpy类型。

    Returns:
        dict (str: np.array) -> tokenized_output = {
                            'input_ids': [[1, 47, 10, 7, 304, 3, 3, 3, 3, 47, 27,
                                           247, 98, 105, 512, 777, 15, 12043, 2], ...],
                            'token_type_ids': [[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                                                0, 0, 0, 0, 0, 0, 0, 0], ...],
                            'attention_mask': [[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                                                1, 1, 1, 1, 1, 1, 1, 1], ...],
                            'mask_positions': [[5, 6, 7, 8], ...],
                            'mask_labels': [[2372, 3442, 0, 0],
                                            [2643, 4434, 2334, 0], ...]
                        }
    """
    tokenized_output = {
        'input_ids': [],
        'token_type_ids': [],
        'attention_mask': [],
        'mask_positions': [],
        'mask_labels': []
    }

    for i, example in enumerate(examples['text']):
        if train_mode:
            label, content = example.strip().split('\t')
        else:
            content = example.strip()

```

```
        inputs_dict = {
            'textA': content,
            'MASK': '[MASK]'
        }
        encoded_inputs = hard_template(
            inputs_dict=inputs_dict,
            tokenizer=tokenizer,
            max_seq_len=max_seq_len,
            mask_length=max_label_len)
        tokenized_output['input_ids'].append(encoded_inputs["input_ids"])
 tokenized_output['token_type_ids'].append(encoded_inputs["token_type_ids"])
 tokenized_output['attention_mask'].append(encoded_inputs["attention_mask"])
 tokenized_output['mask_positions'].append(encoded_inputs["mask_position"])

        if train_mode:
            label_encoded = tokenizer(text=[label])  # 将label补到最大长度
            # print(f'label_encoded-->{label_encoded}')
            label_encoded = label_encoded['input_ids'][0][1:-1]
            label_encoded = label_encoded[:max_label_len]
            add_pad = [tokenizer.pad_token_id] * (max_label_len - len(label_encoded))
            label_encoded = label_encoded + add_pad
            tokenized_output['mask_labels'].append(label_encoded)

    for k, v in tokenized_output.items():
        if return_tensor:
            tokenized_output[k] = torch.LongTensor(v)
        else:
            tokenized_output[k] = np.array(v)

    return tokenized_output


if __name__ == '__main__':
    pc = ProjectConfig()
    train_dataset = load_dataset('text', data_files=pc.train_path)
    print(type(train_dataset))
    print(train_dataset)
    # print('*'*80)
    # print(train_dataset['train']['text'])
    tokenizer = AutoTokenizer.from_pretrained(pc.pre_model)
    hard_template = HardTemplate(prompt='这是一条{MASK}评论：{textA}')

```

```
    convert_func = partial(convert_example,
                           tokenizer=tokenizer,
                           hard_template=hard_template,
                           max_seq_len=30,
                           max_label_len=2)
    dataset = train_dataset.map(convert_func, batched=True)
    for value in dataset['train']:
        print(value)
        print(len(value['input_ids']))
        break

```

3.3 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
from data_preprocess import *
from pet_config import *

pc = ProjectConfig() # 实例化项目配置文件
tokenizer = AutoTokenizer.from_pretrained(pc.pre_model)
```

- 定义获取数据加载器的方法get_data()

```
def get_data():
    # prompt定义
    prompt = open(pc.prompt_file, 'r', encoding='utf8').readlines()
[0].strip()
    hard_template = HardTemplate(prompt=prompt)  # 模板转换器定义
    dataset = load_dataset('text', data_files={'train': pc.train_path,
                                               'dev': pc.dev_path})
    # print(dataset)
    # print(f'Prompt is -> {prompt}')
    new_func = partial(convert_example,
                       tokenizer=tokenizer,
                       hard_template=hard_template,
                       max_seq_len=pc.max_seq_len,
                       max_label_len=pc.max_label_len)

    dataset = dataset.map(new_func, batched=True)

    train_dataset = dataset["train"]
```

```
    dev_dataset = dataset["dev"]
    # print('train_dataset', train_dataset[:2])
    # print('*'*80)
    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

```