基于Zero-shot方式实现LLM文本匹配

学习目标

- 掌握Zero-shot方式下prompt的设计方式
- 掌握利用LLM实现文本匹配的代码

---

1 LLM信息抽取任务介绍

- 首先，我们构造几个短文本对：

```
1.('股票市场今日大涨，投资者乐观。', '持续上涨的市场让投资者感到满意。'),
2.('油价大幅下跌，能源公司面临挑战。', '未来智能城市的建设趋势愈发明显。'),
3.('利率上升，影响房地产市场。', '高利率对房地产有一定冲击。'),
```

- 我们期望模型能够帮我们识别出这 3 对句子中，哪几对描述的是相似的语言。
- 我们期望模型输出的结果为：

```
['相似', '不相似', '相似']
```

2 Prompt设计

- 在该任务的 prompt 设计中，我们主要考虑 2 点：
  - 需要向模型解释什么叫作「文本匹配任务」
  - 需要让模型按照我们指定的格式输出
- 为了让模型知道什么叫做「文本匹配任务」，我们借用 Incontext Learning 的方式，先给模型展示几个正确的例子：

```
>>> User: 句子一: 公司ABC发布了季度财报，显示盈利增长。\n句子二: 财报披露，公司ABC利润上升
>>> Bot: 是
>>> User:句子一: 黄金价格下跌，投资者抛售。\n句子二: 外汇市场交易额创下新高
>>> Bot: 不是
...
```

> 其中，`User` 代表我们输入给模型的句子，`Bot` 代表模型的回复内容。
>
> 注意：上述例子中 `Bot` 的部分也是由人工输入的，其目的是希望看到在看到类似 `User` 中的句子时，模型应当做出类似 `Bot` 的回答。

3 文本匹配任务代码实现

- 本章节使用的模型为ChatGLM-6B，参数参数较大（6B），下载到本地大概需要 12G+ 的磁盘空间，请确保磁盘有充足的空间。此外，加载模型大概需要 13G 左右的显存，如果您显存不够，可以进行模型量化加载以缩小模型成本。
- 本次文本匹配任务实现的主要过程：

    - 构造prompt
    - 实现文本匹配
- 代码存放位置：/Users/***/PycharmProjects/llm/zero-shot/llm_text_matching.py
- llm_information_extraction.py脚本中包含三个函数：init_prompts()和inference()

3.1 导入必备的工具包

```
from rich import print
from transformers import AutoTokenizer, AutoModel

import os


# 提供相似，不相似的语义匹配例子
examples = {
    '是': [
        ('公司ABC发布了季度财报，显示盈利增长。', '财报披露，公司ABC利润上升。'),
    ],
    '不是': [
        ('黄金价格下跌，投资者抛售。', '外汇市场交易额创下新高。'),
        ('央行降息，刺激经济增长。', '新能源技术的创新。')
    ]
}

```

3.2 构建init_prompts()函数

- 目的：进行prompt设计
- 具体代码实现：

```
def init_prompts():
    """
    初始化前置prompt，便于模型做 incontext learning。
    """
    pre_history = [
        (
            '现在你需要帮助我完成文本匹配任务，当我给你两个句子时，你需要回答我这两句话语义是否相似。只需要回答是否相似，不要做多余的回答。',
            '好的，我将只回答”是“或”不是“。'
        )
    ]

    for key, sentence_pairs in examples.items():
        for sentence_pair in sentence_pairs:
            sentence1, sentence2 = sentence_pair
            pre_history.append((
                f'句子一：{sentence1}\n句子二：{sentence2}\n上面两句话是相似的语义吗？',
                key
            ))
```

```
    return {'pre_history': pre_history}
```

3.3 构建inference()函数

- 目的：模型实现信息匹配
- 具体代码实现

```
def inference(
        sentence_pairs: list,
        custom_settings: dict
    ):
    """
    推理函数。

    Args:
        model (transformers.AutoModel): Language Model 模型。
        sentence_pairs (List[str]): 待推理的句子对。
        custom_settings (dict): 初始设定，包含人为给定的 few-shot example。
    """
    for sentence_pair in sentence_pairs:
        sentence1, sentence2 = sentence_pair
        sentence_with_prompt = f'句子一: {sentence1}\n句子二: {sentence2}\n上面两句话是相似的语义吗? '
        response, history = model.chat(tokenizer, sentence_with_prompt, history=custom_settings['pre_history'])
        print(f'>>> [bold bright_red]sentence: {sentence_pair}')
        print(f'>>> [bold bright_green]inference answer: {response}')
        # print(history)
```

- 代码调用

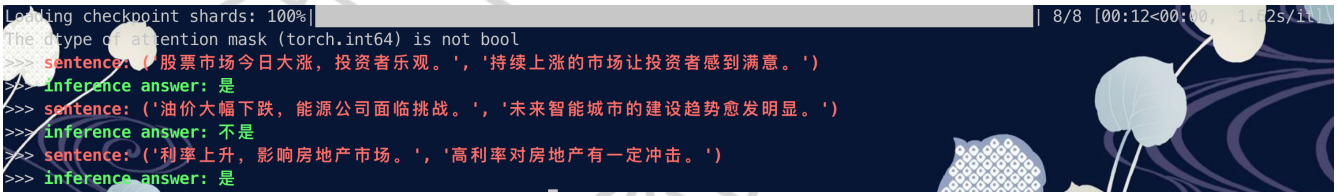
```
if __name__ == '__main__':
    #device = 'cuda:0'
    device = 'cpu'
    tokenizer = AutoTokenizer.from_pretrained("./ChatGLM-6B/THUDM/chatglm-6b",
                                              trust_remote_code=True)
    #model = AutoModel.from_pretrained("./ChatGLM-6B/THUDM/chatglm-6b",
                                      # trust_remote_code=True).half().cuda()
    model = AutoModel.from_pretrained("./ChatGLM-6B/THUDM/chatglm-6b",
                                      trust_remote_code=True).float()
    model.to(device)

    sentence_pairs = [
        ('股票市场今日大涨，投资者乐观。', '持续上涨的市场让投资者感到满意。'),
        ('油价大幅下跌，能源公司面临挑战。', '未来智能城市的建设趋势愈发明显。'),
        ('利率上升，影响房地产市场。', '高利率对房地产有一定冲击。'),
    ]

    custom_settings = init_prompts()
    inference(
        sentence_pairs,
```

```
        custom_settings
    )
```

- 打印结果：



小结总结

本章节主要介绍了如何利用Few-Shot方式基于ChatGLM-6B实现文本匹配任务。