# 知识工程-作业8 中文事件抽取

2024214500 叶璨铭

## 代码与文档格式说明

本文档使用Jupyter Notebook编写，遵循Diátaxis 系统 Notebook实践 <https://nbdev.fast.ai/tutorials/best_practices.html，所以同时包括了实验文档和实验代码>。

本文档理论上支持多个格式，包括ipynb, docx, pdf 等。您在阅读本文档时，可以选择您喜欢的格式来进行阅读，建议您使用 Visual Studio Code (或者其他支持jupyter notebook的IDE, 但是VSCode阅读体验最佳) 打开 ipynb格式的文档来进行阅读。

为了记录我们自己修改了哪些地方，使用git进行版本控制，这样可以清晰地看出我们基于助教的代码在哪些位置进行了修改，有些修改是实现了要求的作业功能，而有些代码是对原本代码进行了重构和优化。我将我在知识工程课程的代码，在作业截止DDL之后，开源到 <https://github.com/2catycm/THU-Coursework-Knowledge-Engineering.git> ，方便各位同学一起学习讨论。

## 代码规范说明

在我们实现函数过程中，函数的docstring应当遵循fastai规范而不是numpy规范，这样简洁清晰，不会Repeat yourself。相应的哲学和具体区别可以看 <https://nbdev.fast.ai/tutorials/best_practices.html#keep-docstrings-short-elaborate-in-separate-cells>

为了让代码清晰规范，在作业开始前，使用 ruff format格式化助教老师给的代码;

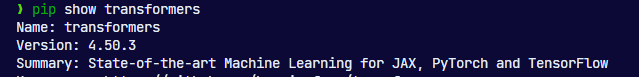
alt text

很好，这次代码格式化没有报错。

注意pylance报错



参考 <https://github.com/huggingface/transformers/issues/15497>



我们的更新。

没区别，实际上就用torch的好了

from transformers import get\_linear\_schedule\_with\_warmup  
from torch.optim import AdamW

## 实验环境准备

采用上次的作业专属环境，为了跑通最新方法，使用3.12 和 torch 2.6

conda create -n assignments python=3.12  
conda activate assignments  
pip install -r ../requirements.txt  
pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu126  
pip install -U git+https://github.com/TorchRWKV/flash-linear-attention

注意到

from seqeval.metrics import precision\_score, recall\_score, f1\_score

参考 <https://github.com/chakki-works/seqeval>

pip install seqeval

## 原理回顾和课件复习

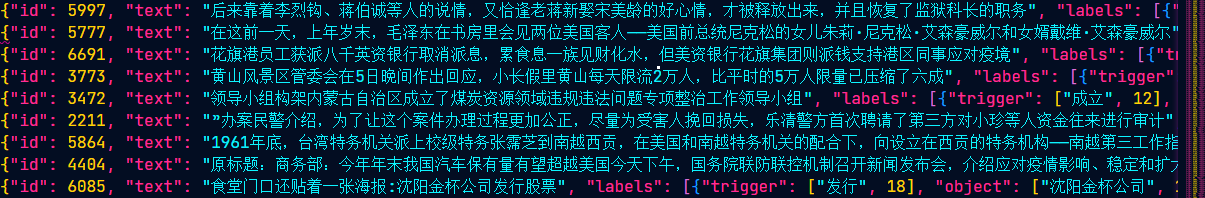
课上详细介绍了事件抽取任务的一些基本特点和难点，事件肯定和时间有关系，是动态的知识， “清华大学在北京”就是相对静态的。“五月份清华举行校庆”那才有事件。实体图谱就没有事件图谱厉害。

事件会有**触发词 trigger**，决定事件的类别；论元 argument，实体、事件、属性，应该有时间，没有事件就不是论元。

## 数据准备

download.sh 的清华网盘链接过期了，还好非常好的学长给的压缩包已经处理好了数据 raw。

现在我们写 preprocess

首先观察数据格式 

这个实际上是jsonl，一行一个json

每一行数据，有 id, text, labels, distant\_trigger 四个字段 id就是数据的id，text就是句子，distant\_trigger 是 句子里面 trigger的列表。labels 则是多个对象，一个对象是有 trigger, object, subject, time 和 location。

{  
 "id": 5009,  
 "text": "晚年的毛岸青关心国家大事，关注祖国统一，拥护改革开放，热心支持老少边穷地区建设，多次和夫人邵华、儿子毛新宇重走长征路，到革命老区、到工厂、到农村调研，并以多种形式帮助失学儿童，支持创办了多个青少年爱国主义教育基地",  
 "labels": [  
 {  
 "trigger": ["支持创办", 88],  
 "object": ["儿子毛新宇", 48],  
 "subject": ["多个青少年爱国主义教育基地", 93],  
 "time": "",  
 "location": "",  
 }  
 ],  
 "distant\_trigger": ["帮助", "支持", "拥护", "建设", "开放", "创办", "改革", "关注"],  
}

为什么 labels里面的比 distant\_trigger 的少呢？

远距离触发词（Distant Trigger）是一组与事件相关的关键词或短语，它们可能不是直接的触发词，但可以提供上下文信息，帮助模型更全面地理解事件的语义。这些词通常是通过某种方法（如规则、统计分析或知识库）从文本中提取出来的，用于扩展触发词的语义范围。在标注数据不足的情况下，远距离触发词可以作为一种辅助信息。

现在我们来写要求的函数 ，首先了解 IOB2 格式 <https://en.wikipedia.org/wiki/Inside%E2%80%93outside%E2%80%93beginning_(tagging)> 。

IOB是 inside, outside, beginning 的缩写，也叫作BIO。

IOB2 比较好理解，

Alex B-PER  
is O  
going O  
to O  
Los B-LOC  
Angeles I-LOC  
in O  
California B-LOC

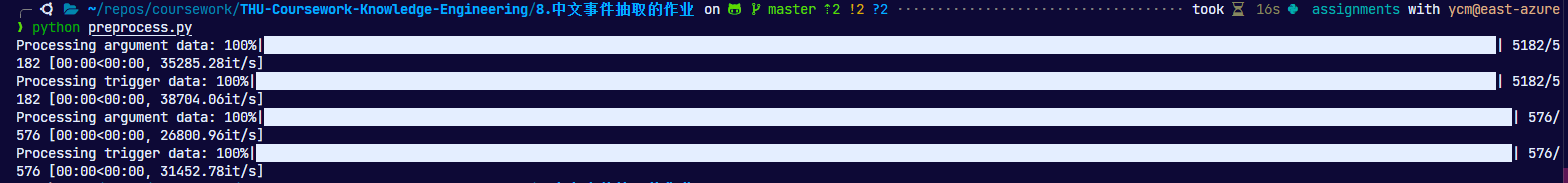
import json  
import os  
from tqdm import tqdm  
def process\_trigger\_data(file):  
 """  
 convert raw data into sequence-labeling format data for trigger identification, and save to `./data/processed/trigger`  
 each line in the converted contains `token[space]label`  
 you can use IOB2 format tagging https://en.wikipedia.org/wiki/Inside%E2%80%93outside%E2%80%93beginning\_(tagging), or other tagging schema you think fit  
 use empty line to indicate end of one sentence  
 if using IOB2 format, the example output would be like  
 ```  
 本 O  
 平 O  
 台 O  
 S O  
 S O  
 R O  
 N O  
 发 B-EVENT  
 表 I-EVENT  
 了 O  
 题 O  
 为 O  
 《 O  
 夏 O  
 季 O  
 冠 O  
 状 O  
 病 O  
 毒 O  
 流 O  
 行 O  
 会 O  
 减 O  
 少 O  
 吗 O  
  
 对 O  
 于 O  
 该 O  
 举 O  
 动 O  
 ```  
 """  
  
 with open(f"./data/raw/{file}.json", encoding="utf-8") as f:  
 lines = f.readlines()  
 outlines = []  
 ##################  
 bar = tqdm(lines, desc="Processing trigger data")  
 for line in bar:  
 sample\_object = json.loads(line.strip())  
 text = sample\_object["text"]  
 tokens = list(text) # 单字分词  
 labels = sample\_object["labels"]  
 labels\_seq = ["O"] \* len(tokens) # 默认情况  
 # 依据 labels 进行标注  
 for label in labels:  
 if label["trigger"]:  
 start = label["trigger"][1] # 起始位置  
 end = start + len(label["trigger"][0]) # 结束位置  
 # 触发词标注  
 labels\_seq[start] = "B-EVENT"  
 for i in range(start + 1, end):  
 labels\_seq[i] = "I-EVENT"  
 # 生成输出  
 for i in range(len(tokens)):  
 outlines.append(f"{tokens[i]} {labels\_seq[i]}")  
 outlines.append("") # 句子结束标志，用空行分割  
  
 ##################  
  
 if not os.path.exists("./data/processed/trigger"):  
 os.makedirs("./data/processed/trigger", exist\_ok=True)  
 with open(f"./data/processed/trigger/{file}.txt", "w") as f:  
 f.writelines("\n".join(outlines))

注意我们忽略了 distant\_trigger 。

同样地来写 process\_argument\_data

def process\_argument\_data(file):  
 """  
 convert raw data into sequence-labeling format data for argument identification, and save to `./data/processed/argument`  
 event triggers are surrounded by `<event>` `<event/>` markers  
 each line in the converted contains `token[space]label`  
 you can use BIO format tagging https://en.wikipedia.org/wiki/Inside%E2%80%93outside%E2%80%93beginning\_(tagging), or other tagging schema you think fit  
 use empty line to indicate end of one sentence  
 if using IOB2 format, the example output would be like  
 ```  
 3 B-object  
 0 I-object  
 年 I-object  
 代 I-object  
 <event> O  
 参 O  
 加 O  
 <event/> O  
 中 B-subject  
 共 I-subject  
 中 I-subject  
 央 I-subject  
 的 I-subject  
 特 I-subject  
 种 I-subject  
 领 I-subject  
 导 I-subject  
 工 I-subject  
 作 I-subject  
  
 他 O  
 告 O  
 诉 O  
 新 O  
 京 O  
 报 O  
 记 O  
 者 O  
 ```  
 """  
 with open(f"./data/raw/{file}.json", encoding="utf-8") as f:  
 lines = f.readlines()  
 outlines = []  
 #############  
 bar = tqdm(lines, desc="Processing argument data")  
 for line in bar:  
 sample\_object = json.loads(line.strip())  
 text = sample\_object["text"]  
 tokens = list(text)  
 labels = sample\_object["labels"]  
 labels\_seq = ["O"] \* len(tokens)  
 # 依据 labels 进行标注  
 for label in labels:  
 # 先处理触发词， 如果强行插入新词的话，不太好，我决定后面输出的时候特别操作。  
 if label["trigger"]:  
 start = label["trigger"][1] # 起始位置  
 end = start + len(label["trigger"][0]) # 结束位置  
 # 触发词标注  
 labels\_seq[start] = "B-EVENT"  
 for i in range(start + 1, end-1):  
 labels\_seq[i] = "O"  
 labels\_seq[end-1] = "E-EVENT" # 触发词最后一个字标注为 E-EVENT  
  
 # 处理 object subject  
 if label["object"]:  
 start = label["object"][1]  
 end = start + len(label["object"][0])  
 labels\_seq[start] = "B-object"  
 for i in range(start + 1, end):  
 labels\_seq[i] = "I-object"  
  
 if label["subject"]:  
 start = label["subject"][1]  
 end = start + len(label["subject"][0])  
 labels\_seq[start] = "B-subject"  
 for i in range(start + 1, end):  
 labels\_seq[i] = "I-subject"  
  
 # 生成输出  
 for i in range(len(tokens)):  
 if labels\_seq[i] == "B-EVENT":  
 outlines.append("<event> O")  
 outlines.append(f"{tokens[i]} O")  
 elif labels\_seq[i] == "E-EVENT":  
 outlines.append(f"{tokens[i]} O")  
 outlines.append("<event/> O")  
 else:  
 # 普通情况  
 outlines.append(f"{tokens[i]} {labels\_seq[i]}")  
 outlines.append("") # 句子结束标志，用空行分割

难点是 event 标签比较特殊。



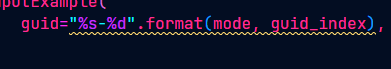
这就成功处理。



“由于一个事件触发词或论元可能跨越多个 token，因此标注格式采用 IOB2” 跨越多个token的意思是多个token组成一个实体，不是说可以跳过中间的token联合后面的token。

## 数据加载

我们看下 utils.py 文件，我们需要完成 read\_examples\_from\_file 和 convert\_examples\_to\_features

首先修复语法错误 

改成 f"{mode:s}-{guid\_index:d}"

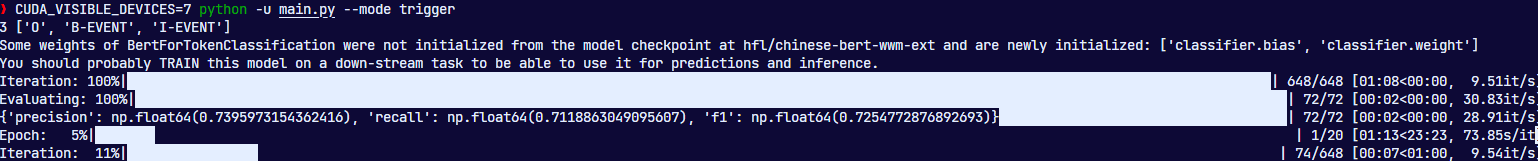
def read\_examples\_from\_file(data\_dir, mode):  
 """  
 read file and convert to a list of `InputExample`s  
 """  
 file\_path = os.path.join(data\_dir, "{}.txt".format(mode))  
 guid\_index = 1  
 examples = []  
 with open(file\_path, encoding="utf-8") as f:  
 file = f.read()  
 samples = file.split("\n\n")  
 for guid\_index, sample in enumerate(samples):  
 lines = sample.split("\n")  
 words = []  
 labels = []  
 for line in lines:  
 if line.strip() == "":  
 continue  
 line = line.split(" ")  
 if len(line) == 2:  
 words.append(line[0])  
 labels.append(line[1])  
 else:   
 raise ValueError(  
 "Error in line format: {} in file {}".format(line, file\_path)  
 )  
  
 examples.append(  
 InputExample(guid=f"{mode:s}-{guid\_index:d}", words=words, labels=labels)  
 )  
 return examples

原本的逻辑不太方便，我直接重构了。

def convert\_examples\_to\_features(  
 examples: List[InputExample],  
 label\_list, # a list of all unique labels  
 max\_seq\_length: int, # all sequence should be padded or truncated to `max\_seq\_length`  
 tokenizer, # PretrainedTokenizer  
 pad\_token\_label\_id: int, # label id for pad token  
) -> List[InputFeatures]: # features  
 """Loads a list of `InputExample`s into a list of `InputFeatures`s"""  
 cls\_token = tokenizer.cls\_token  
 sep\_token = tokenizer.sep\_token  
 pad\_token\_id = tokenizer.pad\_token\_id # padded token id  
 # tokenizer.convert\_tokens\_to\_ids([tokenizer.pad\_token])[0]  
 label\_map = {label: i for i, label in enumerate(label\_list)}  
  
 features = []  
 for ex\_index, example in enumerate(examples):  
 # Hint: remember to add `[CLS]` and `[SEP]` tokens for BERT model  
 # e.g. [CLS] the dog is hairy . [SEP]  
 tokens = []  
 label\_ids = []  
 for word, label in zip(example.words, example.labels):  
 word\_tokens = tokenizer.tokenize(word)  
  
 # Bert模型中，一个单词可能会被切分成多个子词，我们需要将标签分配给这些子词  
 if len(word\_tokens) > 0:  
 tokens.extend(word\_tokens)  
 # 只有第一个子词保留原始标签，其余子词使用特殊标签 "X"  
 label\_ids.extend([label\_map[label]] + [pad\_token\_label\_id] \* (len(word\_tokens) - 1))  
  
 # 添加[CLS]和[SEP]标记  
 special\_tokens\_count = 2  
 if len(tokens) > max\_seq\_length - special\_tokens\_count:  
 tokens = tokens[: (max\_seq\_length - special\_tokens\_count)]  
 label\_ids = label\_ids[: (max\_seq\_length - special\_tokens\_count)]  
  
 tokens += [sep\_token]  
 label\_ids += [pad\_token\_label\_id]  
 tokens = [cls\_token] + tokens  
 label\_ids = [pad\_token\_label\_id] + label\_ids  
  
 input\_ids = tokenizer.convert\_tokens\_to\_ids(tokens)  
  
 # 注意mask的处理，只有真实的token对应的mask值为1，padding的token对应的mask值为0  
 input\_mask = [1] \* len(input\_ids)  
 padding\_length = max\_seq\_length - len(input\_ids)  
 input\_ids = input\_ids + ([pad\_token\_id] \* padding\_length)  
 input\_mask = input\_mask + ([0] \* padding\_length)  
 label\_ids = label\_ids + ([pad\_token\_label\_id] \* padding\_length)  
  
 assert len(input\_ids) == max\_seq\_length  
 assert len(input\_mask) == max\_seq\_length  
 assert len(label\_ids) == max\_seq\_length  
  
 features.append(  
 InputFeatures(  
 input\_ids=input\_ids, input\_mask=input\_mask, label\_ids=label\_ids  
 )  
 )  
 return features

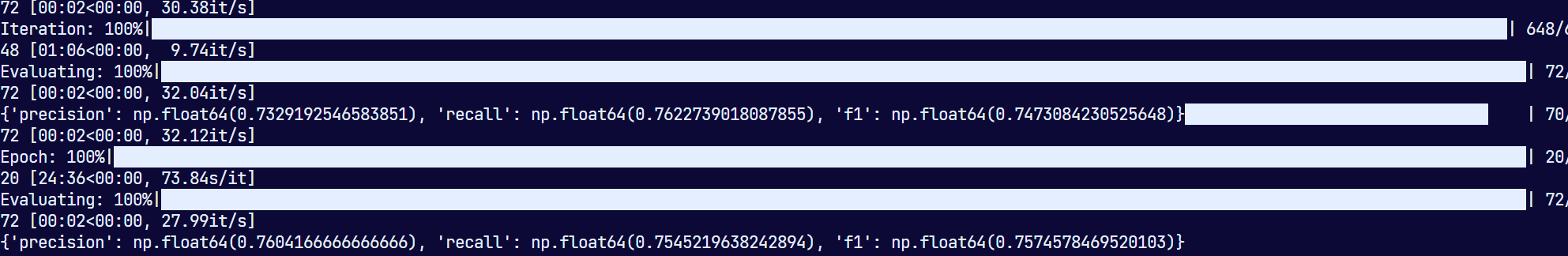
## 运行效果

CUDA\_VISIBLE\_DEVICES=7 python -u main.py --mode trigger



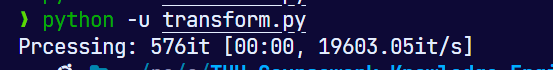
首先下载了 tokenizer

然后开始训练。



然后我们运行

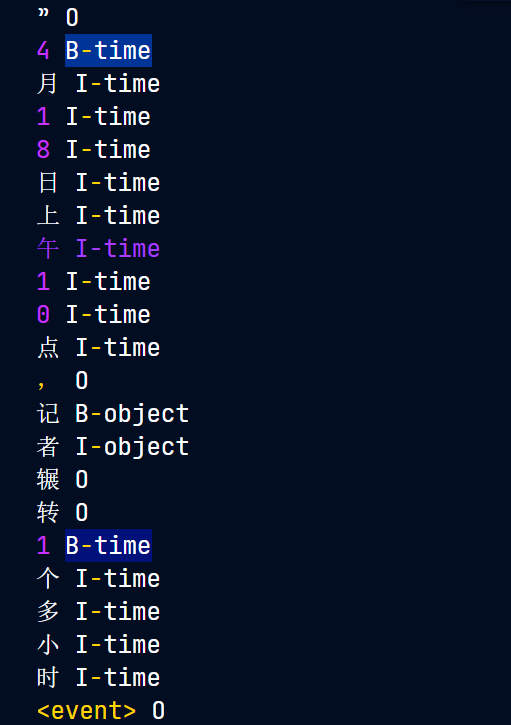
python -u transform.py

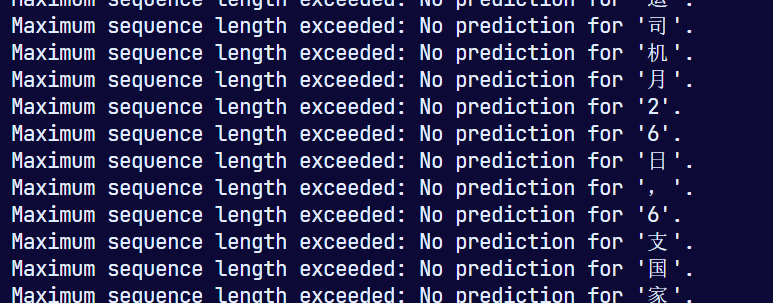


python -u main.py --mode argument

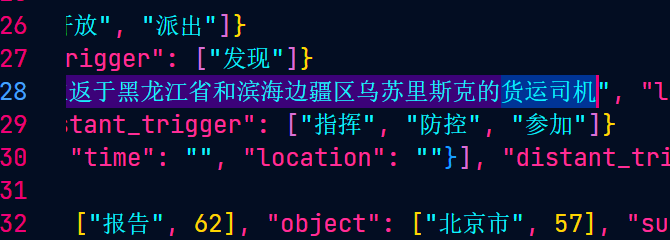
注意学长 README 忘记 --mode 了

运行到最后我们才发现，test.txt 里面有 B-time 和 B-location





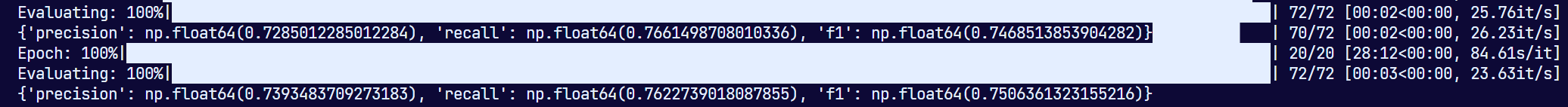
还遇到了 Max seq length的警告。 这是正常的，代码中确实模型设置了 max length 而数据集中确实有“货运司机”的数据，是在句子的末尾，所以无法预测，相应地评估也不评估就行了。



紧急修复代码

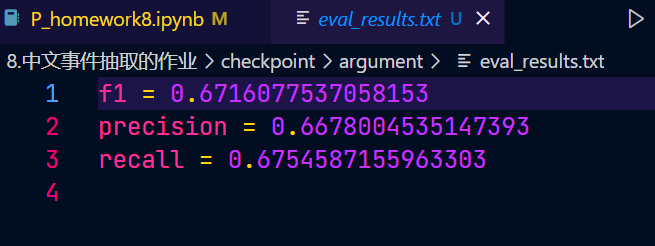
if label['time']:  
 start = label['time'][1]  
 end = start + len(label['time'][0])  
 labels\_seq[start] = "B-time"  
 for i in range(start + 1, end):  
 labels\_seq[i] = "I-time"  
  
if label['location']:  
 start = label['location'][1]  
 end = start + len(label['location'][0])  
 labels\_seq[start] = "B-location"  
 for i in range(start + 1, end):  
 labels\_seq[i] = "I-location"

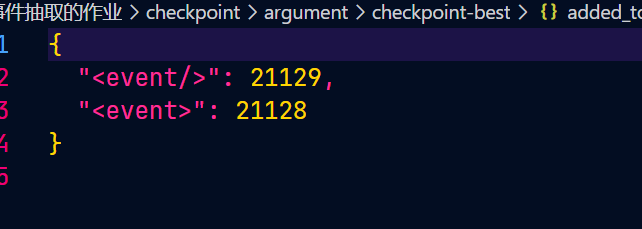
 重新运行。

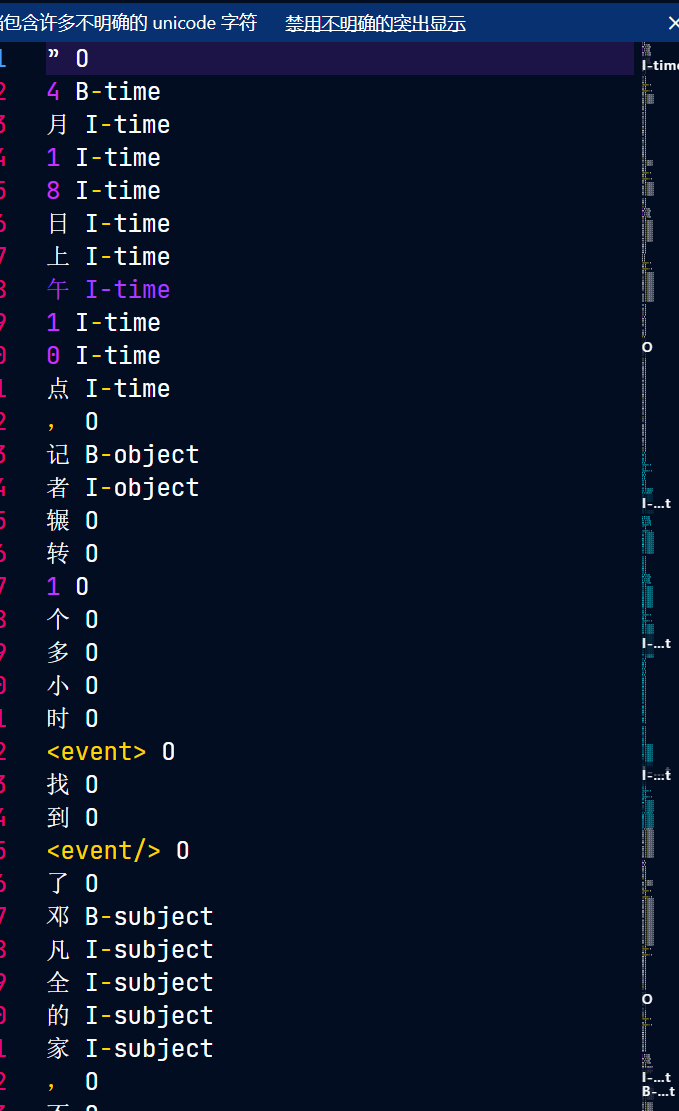
 一阶段的结果是一样的。

再次运行

这一次我们运行成功，可以查看结果



注意tokenizer里面自动增加了两个新的token 



预测的输出看起来非常合理。