

基于ClueAI/PromptCLUE模型进行文本学习

石常乐 221017000223

陈碧萱 221017000231

模型简述：

支持不同类型的任务。针对理解类任务，可以自定义标签体系；针对生成任务，可以进行采样自由生成。具有更好的理解、生成和抽取能力，并且支持文本改写、纠错、知识图谱问答。实现了中文上的三大统一：统一模型框架，统一任务形式，统一应用方式。

目的：

通过该模型的搭建，测试得出结论以及解决在过程中遇见的问题进一步了解文本挖掘的过程及目的。

环境搭建过程

机器规格：

GPU环境 ⓘ

剩余额度 ⓘ 30小时19分钟

- 8核 32GB 显存24G
- 预装 ModelScope Library
- 预装镜像

ubuntu20.04-cuda11.8.0-py38-torch2.0.1-tf2.13.0-1.9.5

安装transformer等模块：

```
!pip install sentencepiece
!pip install transformers
!pip install torch
!pip install rich[jupyter]
```

Looking in indexes: <https://mirrors.aliyun.com/pypi/simple>
Requirement already satisfied: sentencepiece in /opt/conda/lib/python3.8/site-packages (0.1.99)
DEPRECATION: pytorch-lightning 1.7.7 has a non-standard dependency specifier torch>=1.9.*. pip 23.3 will enforce this behavior change. A possible replacement is to upgrade to a newer version of pytorch-lightning or contact the author to suggest that they release a version with a conforming dependency specifiers. Discussion can be found at <https://github.com/pytorch/pytorch/issues/12063>
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: <https://pip.pypa.io/warnings/venv>
Looking in indexes: <https://mirrors.aliyun.com/pypi/simple>
Requirement already satisfied: transformers in /opt/conda/lib/python3.8/site-packages (4.34.1)
Requirement already satisfied: filelock in /opt/conda/lib/python3.8/site-packages (from transformers) (3.12.2)
Requirement already satisfied: huggingface-hub<1.0,>=0.16.4 in /opt/conda/lib/python3.8/site-packages (from transformers) (0.17.3)
Requirement already satisfied: numpy>=1.17 in /opt/conda/lib/python3.8/site-packages (from transformers) (1.24.3)
Requirement already satisfied: packaging>=20.0 in /opt/conda/lib/python3.8/site-packages (from transformers) (23.0)
Requirement already satisfied: pyyaml>=5.1 in /opt/conda/lib/python3.8/site-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /opt/conda/lib/python3.8/site-packages (from transformers) (2023.8.8)
Requirement already satisfied: requests in /opt/conda/lib/python3.8/site-packages (from transformers) (2.31.0)
Requirement already satisfied: tokenizers<0.15,>=0.14 in /opt/conda/lib/python3.8/site-packages (from transformers) (0.14.1)
Requirement already satisfied: safetensors>=0.3.1 in /opt/conda/lib/python3.8/site-packages (from transformers) (0.3.3)
Requirement already satisfied: tqdm>=4.27 in /opt/conda/lib/python3.8/site-packages (from transformers) (4.65.0)
Requirement already satisfied: fsspec in /opt/conda/lib/python3.8/site-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (2023.10.0)

引入基础依赖，包含系统，时间以及模型依赖等，用于基础环境配置：

```
import os,json
import numpy as np
import pandas as pd
import torch
import torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader, RandomSampler, SequentialSampler
import os,time
# Importing the T5 modules from huggingface/transformers
from transformers import T5Tokenizer, T5ForConditionalGeneration

# rich: for a better display on terminal
from rich.table import Table
from rich import box
from rich.console import Console
print("end2...")
```

end2...

数据准备

因为通过文本进行自主学习，所以需要大量数据。下载训练数据：

```
!wget https://raw.githubusercontent.com/CLUEbenchmark/pCLUE/main/datasets/pCLUE_train_1.json
!wget https://raw.githubusercontent.com/CLUEbenchmark/pCLUE/main/datasets/pCLUE_train_2.json
!wget https://raw.githubusercontent.com/CLUEbenchmark/pCLUE/main/datasets/pCLUE_train_3.json
!wget https://raw.githubusercontent.com/CLUEbenchmark/pCLUE/main/datasets/pCLUE_train_4.json
!wget https://raw.githubusercontent.com/CLUEbenchmark/pCLUE/main/datasets/pCLUE_train_5.json
!wget https://raw.githubusercontent.com/CLUEbenchmark/pCLUE/main/datasets/pCLUE_train_6.json
!wget https://raw.githubusercontent.com/CLUEbenchmark/pCLUE/main/datasets/pCLUE_train_7.json
!wget https://raw.githubusercontent.com/CLUEbenchmark/pCLUE/main/datasets/pCLUE_train_8.json
!wget https://raw.githubusercontent.com/CLUEbenchmark/pCLUE/main/datasets/pCLUE_train_9.json

200 OK
长度: 99844693 (95M) [text/plain]
正在保存至: "pCLUE_train_7.json"

pCLUE_train_7.json 100%[=====>] 95.22M 32.2MB/s 用时 3.0s

2023-12-10 07:03:25 (32.2 MB/s) - 已保存 "pCLUE_train_7.json" [99844693/99844693])

--2023-12-10 07:03:25-- https://raw.githubusercontent.com/CLUEbenchmark/pCLUE/main/datasets/pCLUE_train_8.json
正在解析主机 raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
正在连接 raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... 已连接。
已发出 HTTP 请求, 正在等待回应...
```

查看下载数据:

```
!wc -l pCLUE_train.json

675025 pCLUE_train.json
```

json格式转换为csv:

```
# 数据准备: 将json文件转化为csv形式的文件。
def convert_json_to_csv(source_file, target_file):
    """将json文件转化为csv形式的文件。
    source_file:输入文件;
    target_file: 转化后的文件
    """
    lines=open(source_file,'r').readlines()
    print("length of lines:",len(lines))
    input_list=[]
    output_list=[]
    answer_choices_list=[]
    type_list=[]
    for i, line in enumerate(lines):
        # {"input": " 以下内容是真: "潞县地区专员张友道说:大都架到高处了"那么下面的陈述: "张友道对身边的官员说了话。"是真的,假的,或未知? \n答案: "
        # 1)获得字段值
        json_string=json.loads(line.strip())
        input_=json_string["input"].replace("\n", "_")
        output_=json_string["target"]
        answer_choices_=json_string.get("answer_choices",[])
        type_=json_string["type"]
        if i<10:print(i,"input:",input_,"output:",output_)
        # 2)添加到列表中
        input_list.append(input_)
        output_list.append(output_)
        answer_choices_list.append(answer_choices_)
        type_list.append(type_)

    # 3)写成pandas的dataframe, 以csv进行保存
    df = pd.DataFrame({'input': input_list,
                       'target':output_list,
                       'answer_choices': answer_choices_list,
                       'type': type_list,
```

转换后的数据:


```

):
    """
    Initializes a Dataset class

    Args:
        dataframe (pandas.DataFrame): Input dataframe
        tokenizer (transformers.tokenizer): Transformers tokenizer
        source_len (int): Max length of source text
        target_len (int): Max length of target text
        source_text (str): column name of source text
        target_text (str): column name of target text
    """
    self.tokenizer = tokenizer
    self.data = dataframe
    self.source_len = source_len
    self.summ_len = target_len
    self.target_text = self.data[target_text]
    self.source_text = self.data[source_text]

def __len__(self):
    """returns the length of dataframe"""

    return len(self.target_text)

def __getitem__(self, index):
    """return the input ids, attention masks and target ids"""

    source_text = str(self.source_text[index])
    target_text = str(self.target_text[index])

    # cleaning data so as to ensure data is in string type
    source_text = " ".join(source_text.split())
    target_text = " ".join(target_text.split())

    source = self.tokenizer.batch_encode_plus(
        [source_text],
        max_length=self.source_len,
        pad_to_max_length=True,
        truncation=True,
        padding="max_length",
        return_tensors="pt",
    )
    target = self.tokenizer.batch_encode_plus(
        [target_text],
        max_length=self.summ_len,
        pad_to_max_length=True,
        truncation=True,
        padding="max_length",
        return_tensors="pt",
    )

    source_ids = source["input_ids"].squeeze()

```

```

source_mask = source["attention_mask"].squeeze()
target_ids = target["input_ids"].squeeze()
target_mask = target["attention_mask"].squeeze()

return {
    "source_ids": source_ids.to(dtype=torch.long),
    "source_mask": source_mask.to(dtype=torch.long),
    "target_ids": target_ids.to(dtype=torch.long),
    "target_ids_y": target_ids.to(dtype=torch.long),
}
print("end...")

```

定义训练方法：

```

def train(epoch, tokenizer, model, device, loader, optimizer):

    """
    用于训练的方法
    Function to be called for training with the parameters passed from main function
    """

    print(model);
    model.train()
    time1=time.time()
    for _, data in enumerate(loader, 0):
        y = data["target_ids"].to(device, dtype=torch.long)
        y_ids = y[:, :-1].contiguous() # target, from start to end(except end of token,
<EOS>). e.g. "你好吗? "
        lm_labels = y[:, 1:].clone().detach() # target, for second to end.e.g."好吗?
<EOS>"

        lm_labels[y[:, 1:] == tokenizer.pad_token_id] = -100 # releted to pad_token and
loss. for detail, check here: https://github.com/Shivanandroy/T5-Finetuning-
PyTorch/issues/3
        ids = data["source_ids"].to(device, dtype=torch.long) # input. e.g. "how are
you?"

        mask = data["source_mask"].to(device, dtype=torch.long)

        outputs = model(
            input_ids=ids,
            attention_mask=mask,
            decoder_input_ids=y_ids,
            labels=lm_labels,
        )
        loss = outputs[0]
        # 每100步打印日志
        if _ % 100 == 0 and _!=0:
            time2=time.time()
            print(_, "epoch:"+str(epoch)+"-loss:"+str(loss)+"each step's time
spent:"+str(float(time2-time1)/float(_+0.0001)))
            # training_logger.add_row(str(epoch), str(_), str(loss))
            # console.print(training_logger)

```

```

optimizer.zero_grad()
loss.backward()
optimizer.step()

print("end...")

```

定义验证方法：

```

def validate(epoch, tokenizer, model, device, loader,max_length):
    """
    用于验证的方法：输入用于验证的数据，返回模型预测的结果和正确的标签
    Function to evaluate model for predictions
    """
    model.eval()
    predictions = []
    actuals = []
    with torch.no_grad():
        for _, data in enumerate(loader, 0):
            y = data['target_ids'].to(device, dtype = torch.long)
            ids = data['source_ids'].to(device, dtype = torch.long)
            mask = data['source_mask'].to(device, dtype = torch.long)

            generated_ids = model.generate(
                input_ids = ids,
                attention_mask = mask,
                max_length=max_length,
                num_beams=2,
                repetition_penalty=2.5,
                length_penalty=1.0,
                early_stopping=True
            )
            preds = [tokenizer.decode(g, skip_special_tokens=True,
clean_up_tokenization_spaces=True) for g in generated_ids]
            target = [tokenizer.decode(t, skip_special_tokens=True,
clean_up_tokenization_spaces=True)for t in y]
            if _%1000==0:
                console.print(f'Completed {_}')
            predictions.extend(preds)
            actuals.extend(target)
    return predictions, actuals
print("end...")

```

定义训练类：

```

# 训练类：整合数据集类、训练方法、验证方法，加载数据进行训练并验证训练过程的效果
def T5Trainer(
    dataframe, source_text, target_text, model_params, output_dir="./outputs/"
):
    """
    T5 trainer
    """

```

```

# Set random seeds and deterministic pytorch for reproducibility
torch.manual_seed(model_params["SEED"]) # pytorch random seed
np.random.seed(model_params["SEED"]) # numpy random seed
torch.backends.cudnn.deterministic = True

# logging
console.log(f"[Model]: Loading {model_params["MODEL"]}...\n")

# tokenizer for encoding the text
tokenizer = T5Tokenizer.from_pretrained(model_params["MODEL"])

# Defining the model. We are using PromptCLUE model and added a Language model layer
on top for generation of prediction.
# Further this model is sent to device (GPU/TPU) for using the hardware.
model = T5ForConditionalGeneration.from_pretrained(model_params["MODEL"])
model = model.to(device)

# logging
console.log(f"[Data]: Reading data...\n")

# Importing the raw dataset
dataframe = dataframe[[source_text, target_text]]
# display_df(dataframe.head(2))

# Creation of Dataset and Dataloader
# Defining the train size So 94% of the data will be used for training and the rest
for validation.
train_size = 0.94
train_dataset = dataframe.sample(frac=train_size, random_state=model_params["SEED"])
val_dataset = dataframe.drop(train_dataset.index).reset_index(drop=True)
train_dataset = train_dataset.reset_index(drop=True)

# 打印数据集相关日志：数据量、训练步数
console.print(f"FULL Dataset: {dataframe.shape}")
console.print(f"TRAIN Dataset: {train_dataset.shape}")
console.print(f"TEST Dataset: {val_dataset.shape}\n")
total_train_steps=int((train_dataset.shape[0] *
model_params["TRAIN_EPOCHS"])/model_params["TRAIN_BATCH_SIZE"])
console.print(f"Total Train Steps: {total_train_steps}\n")

# Creating the Training and Validation dataset for further creation of Dataloader
training_set = YourDataSetClass(
    train_dataset,
    tokenizer,
    model_params["MAX_SOURCE_TEXT_LENGTH"],
    model_params["MAX_TARGET_TEXT_LENGTH"],
    source_text,
    target_text,
)
val_set = YourDataSetClass(
    val_dataset,
    tokenizer,

```



```

        model_params["MAX_SOURCE_TEXT_LENGTH"],
        model_params["MAX_TARGET_TEXT_LENGTH"],
        source_text,
        target_text,
    )

# Defining the parameters for creation of dataloaders
train_params = {
    "batch_size": model_params["TRAIN_BATCH_SIZE"],
    "shuffle": True,
    "num_workers": 0,
}

val_params = {
    "batch_size": model_params["VALID_BATCH_SIZE"],
    "shuffle": False,
    "num_workers": 0,
}

# Creation of Dataloaders for testing and validation. This will be used down for
training and validation stage for the model.
training_loader = DataLoader(training_set, **train_params)
val_loader = DataLoader(val_set, **val_params)

# Defining the optimizer that will be used to tune the weights of the network in the
training session.
optimizer = torch.optim.Adam(
    params=model.parameters(), lr=model_params["LEARNING_RATE"]
)

# Training loop
console.log(f"[Initiating Fine Tuning]...\n")

for epoch in range(model_params["TRAIN_EPOCHS"]):
    # 1) train for one epoch
    train(epoch, tokenizer, model, device, training_loader, optimizer)

    # 2) save model for each epoch
    console.log(f"[Saving Model]...\n")
    path = os.path.join(output_dir, "model_files")
    model.save_pretrained(path)
    tokenizer.save_pretrained(path)

    # 3) evaluating test dataset
    console.log(f"[Initiating Validation]...\n")
    with torch.no_grad(): # add 2022.10.4
        #for epoch in range(model_params["VAL_EPOCHS"]):
        predictions, actuals = validate(epoch, tokenizer, model, device,
val_loader, model_params["MAX_TARGET_TEXT_LENGTH"])
        final_df = pd.DataFrame({"Generated Text": predictions, "Actual Text":
actuals})
        final_df.to_csv(os.path.join(output_dir, "predictions.csv"))

```

```

console.save_text(os.path.join(output_dir, "logs.txt"))

console.log(f"[Validation Completed.]\n")
console.print(
    f""[Model] Model saved @ {os.path.join(output_dir, "model_files")}\n""
)
console.print(
    f""[Validation] Generation on Validation data saved @
{os.path.join(output_dir, 'predictions.csv')}\n""
)
console.print(f""[Logs] Logs saved @ {os.path.join(output_dir, 'logs.txt')}\n"")
print("end...")

```

定义模型参数（主要参数控制）：

```

# 定义模型的参数 let's define model parameters specific to T5

model_params = {

    "MODEL": "ClueAI/PromptCLUE", # model_type  ClueAI/PromptCLUE

    "TRAIN_BATCH_SIZE": 8, # training batch size, 8

    "VALID_BATCH_SIZE": 8, # validation batch size,8

    "TRAIN_EPOCHS": 1, # number of training epochs

    "VAL_EPOCHS": 1, # number of validation epochs

    "LEARNING_RATE": 1e-4, # learning rate

    "MAX_SOURCE_TEXT_LENGTH": 512, # max length of source text, 512

    "MAX_TARGET_TEXT_LENGTH": 64, # max length of target text,64

    "SEED": 42, # set seed for reproducibility

}

print("end...")

```

训练模型：

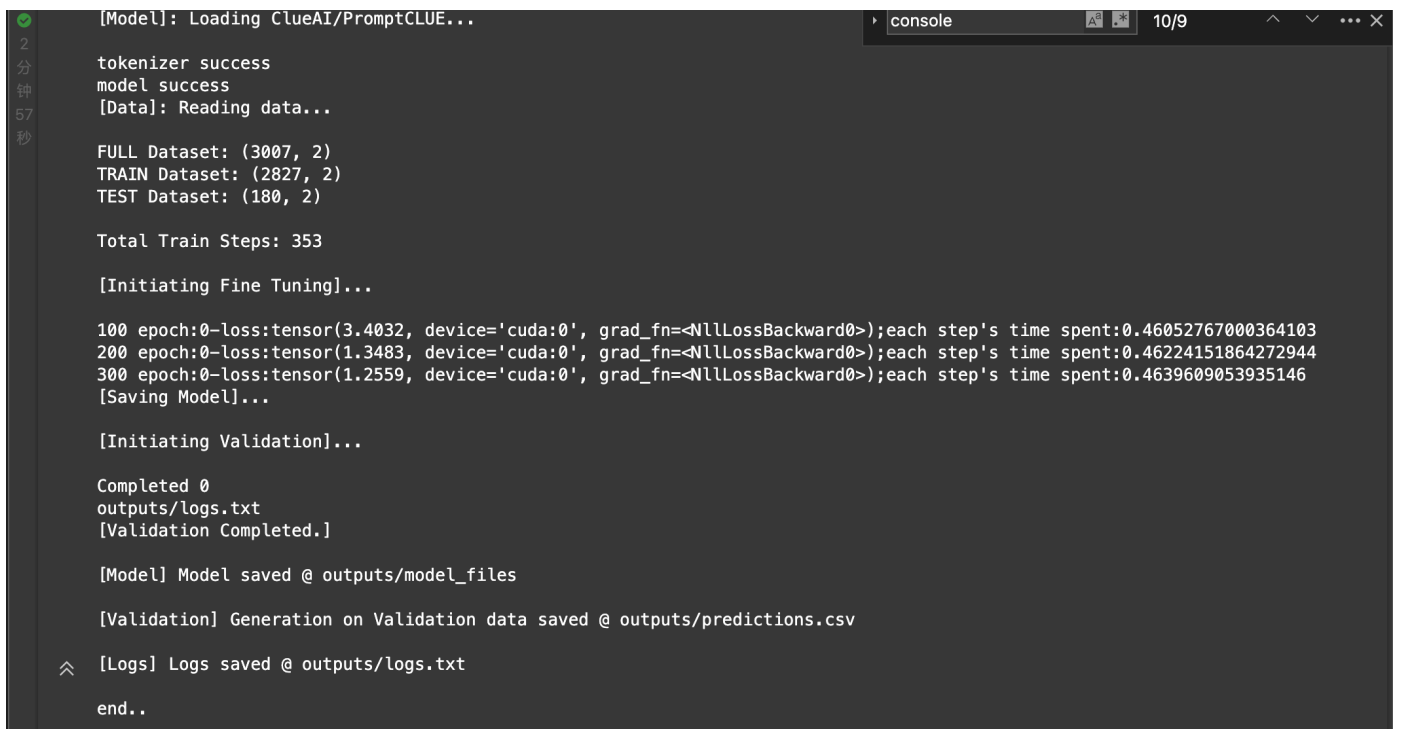
```

# 训练模型
# 使用 pCLUE:1200000+多任务提示学习数据集 的部分数据
# dataframe必须有2列：
#   - input: 文本输入
#   - target: 目标输出
df = pd.read_csv('/content/pCLUE_train.csv') # 数据量：1200k数据。
df = df.sample(frac=0.01) # TODO 取消本行代码，如果你需要更多数据训练

```

```
print("df.head:",df.head(n=5))
print("df.shape:",df.shape)
# 显存占用说明: 如果运行现在显存不足, 请使用nvidia-smi查看显存; 如果显卡多数被占用了, 请重启colab程序
T5Trainer(
    dataframe=df,
    source_text="input",
    target_text="target",
    model_params=model_params,
    output_dir="outputs",
)
print("end..")
```

训练结果:



```
[Model]: Loading ClueAI/PromptCLUE...
tokenizer success
model success
[Data]: Reading data...

FULL Dataset: (3007, 2)
TRAIN Dataset: (2827, 2)
TEST Dataset: (180, 2)

Total Train Steps: 353

[Initiating Fine Tuning]...

100 epoch:0-loss:tensor(3.4032, device='cuda:0', grad_fn=<NllLossBackward0>);each step's time spent:0.46052767000364103
200 epoch:0-loss:tensor(1.3483, device='cuda:0', grad_fn=<NllLossBackward0>);each step's time spent:0.46224151864272944
300 epoch:0-loss:tensor(1.2559, device='cuda:0', grad_fn=<NllLossBackward0>);each step's time spent:0.4639609053935146
[Saving Model]...

[Initiating Validation]...

Completed 0
outputs/logs.txt
[Validation Completed.]

[Model] Model saved @ outputs/model_files

[Validation] Generation on Validation data saved @ outputs/predictions.csv

[Logs] Logs saved @ outputs/logs.txt

end..
```

		Generated Text	Actual Text
1	0	假的	未知
2	1	摄政尚弘才等人奉尚贞王之命	摄政尚弘才等人奉尚贞王之命，以蔡铎为中心，对《中山世鉴》进行汉译和改订
3	2	真的	真的
4	3	未知	假的
5	4	是的	也许
6	5	杜文辉在热身赛中因骨折养伤3个月？	杜文辉什么时候正式进入北京国安三队？
7	6	错误	错误
8	7	不是	不是
9	8	文化	文化
10	9	Microsoft Windows、Mac OS X及Linux作业系统	Microsoft Windows、Mac OS X及Linux作业系统
11	10	是的	也许
12	11	是的	不是
13	12	是的	是的
14	13	是的	不是
15	14	是的	也许
16	15	军事	军事
17	16	娱乐	娱乐
18	17	汽车	汽车
19	18	单刀赴会	千军万马
20	19	绘声绘色	绘声绘色
21	20	未知	真的
22	21	是的	也许

模型验证

```
from transformers import AutoTokenizer, AutoModelForSeq2SeqLM
import torch
import os,time
from transformers import AutoTokenizer

tokenizer =
AutoTokenizer.from_pretrained("/mnt/workspace/.cache/modelscope/ClueAI/PromptCLUE")
model_trained =
AutoModelForSeq2SeqLM.from_pretrained("/mnt/workspace/outputs/model_files/")

# 修改colab笔记本设置为gpu，推理更快
device = torch.device('cpu') # cuda
model_trained.to(device)
def preprocess(text):
    return text.replace("\n", "_")
def postprocess(text):
    return text.replace("_", "\n")

def answer_fn(text, sample=False, top_p=0.6):
    '''sample: 是否抽样。生成任务，可以设置为True;
    top_p: 0-1之间，生成的内容越多样、
    ...
    text = preprocess(text)
    encoding = tokenizer(text=[text], truncation=True, padding=True, max_length=768,
return_tensors="pt").to(device)
    if not sample: # 不进行采样
        out = model_trained.generate(**encoding, return_dict_in_generate=True,
output_scores=False, max_length=128, num_beams=4, length_penalty=0.6)
```

```

else: # 采样 (生成)
    out = model_trained.generate(**encoding, return_dict_in_generate=True,
output_scores=False, max_length=128, do_sample=True, top_p=top_p)
    out_text = tokenizer.batch_decode(out["sequences"], skip_special_tokens=True)
    return postprocess(out_text[0])
print("end...")

```

demo1:

```

text="这是关于哪方面的新闻: 故事,文化,娱乐,体育,财经,房产,汽车,教育,科技,军事,旅游,国际,股票,农业,游戏?如果日本沉没, 中国会接收日本难民吗? "
time1=time.time()
num_times=1
for i in range(num_times):
    result=answer_fn(text, sample=False, top_p=0.6)
    print("result2:",result)
time2=time.time()
time_spent=float(time2-time1)/float(num_times)
print("time spent for single input:"+str(time_spent))

```

结果:

```

X end...
end...
result2: 国际
time spent for single input:0.21890020370483398

```

调整参数,sample=True,结果为: 国际。说明该参数的调整对测试用例结果无区别。

demo2:

```

text="以下两句话是否表达相同意思: \n文本1: 糖尿病腿麻木怎么办? \n文本2: 糖尿病怎样控制生活方式\n选项: 相似, 不相似\n答案: "
result=answer_fn(text, sample=True, top_p=0.9)
print("result2:",result)

result2: 不相似

```

调整文本内容:

```

text="以下两句话是否表达相同意思: \n文本1: 糖尿病腿麻木怎么办? \n文本2: 怎么解决糖尿病间接导致腿麻木\n选项: 相似, 不相似\n答案: "
result=answer_fn(text, sample=True, top_p=0.9)
print("result2:",result)

result2: 相似

```

demo3:

```

text="信息抽取: \n张玄武1990年出生中国国籍无境外居留权博士学位现任杭州线锁科技技术总监.\n问题: 机构, 人名, 职位, 籍贯, 专业, 国籍, 种族\n答案: "
result=answer_fn(text, sample=True, top_p=0.9)
print("result2:",result)

result2: 机构: 杭州线锁科技技术总监
人名: 张玄武
职位: 博士学位, 博士学位

```

demo4:

```
text="翻译成中文: \nThis is a dialogue robot that can talk to people.\n答案: "  
result=answer_fn(text, sample=True, top_p=0.9)  
print("result2:",result)|
```

✕ result2: 这是一部可以对人说话的机器人。

过程中遇到的问题:

- 1, 模型在代码中加载失败, 手动下载模型并通过目录读取。
- 2, 数据读取格式错误, 手动编写读取文件过程。

结果:

通过部分示例, 可以检测出结果大部分正确, 但是也有数据不正确。比如demo4,翻译成俄文, 未翻译成功。

分工:

石常乐 221017000223: 前期资料收集、环境搭建、创建模型、训练模型、发现并调整文本学习过程中所以到问题等

陈碧萱 221017000231: 环境搭建、数据准备、验证模型训练成果、发现并调整文本学习过程中所以到问题、后期资料整理等