Simple_RAG_Project_Implementation

Author: Williams JIANG

任务一：simple RAG 实现

```
任务简报：
非一般程序员训练营 第二季 —— RAG 潘多拉宝盒 任务一】
任务名称：simple RAG 实现
参考资料：
1.大模型（LLMs）simple_RAG 实现篇 大模型（LLMs）simple_RAG 实现篇

任务说明：先把这个项目跑起来，把 基本逻辑搞懂，然后后面再一步步优化
任务截止时间：20240502（本周四）下午8点
任务打卡方式：本次活动采用【知识星球】打卡方式，即将个人学习文档（包含：项目运行效果、基
未打卡惩罚：送飞机票一张（退出本次学习活动）
本次活动最后将根据知识星球打卡榜评选出前三名，并送出精美小礼物！！
```

https://articles.zsxq.com/id_roqf2ge4c1b3.html

使用HuggingFace的transformers库和Meta的FAISS来实现RAG

本地部署模型

安装依赖：

新建环境ragPeoject

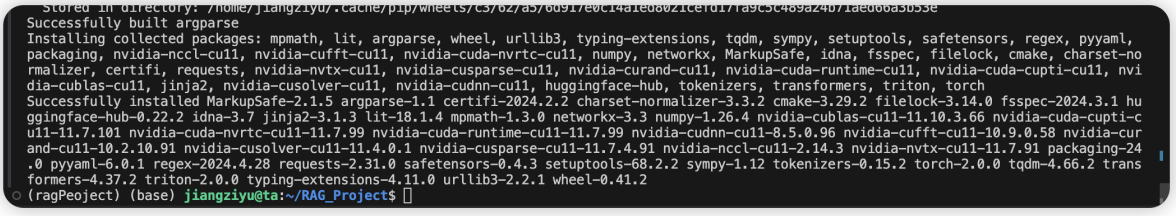
1.2 requirment

必需项	至少	推荐
python	3.8	3.10
torch	1.13.1	2.2.0
transformers	4.37.2	4.38.2
faiss	1.7.2	1.7.2
argparse	1.1	1.1
peft	0.9.0	0.9.0
trl	0.7.11	0.7.11

可选项	至少	推荐
CUDA	11.6	12.2
deepspeed	0.10.0	0.13.1
bitsandbytes	0.39.0	0.41.3
flash-attn	2.3.0	2.5.5

在安装中，faiss 1.7.2通过离线安装了faiss-gpu==1.7.2

通过requirements文件安装了其他依赖



安装成功

Embedding下载，使用 BAAI/bge-large-zh-v1.5 这个Embedding 模型

使用huggingface_cli 通过镜像加速下载bge embedding模型

代码实现

将所有预处理和相关工具类都放入rag_utils这个工具类中以便调用

RAG_Utils 类：

```
import os
import argparse
import faiss
from transformers import AutoTokenizer, AutoModel, AutoModelForCausalLM
import torch
from tqdm import tqdm

class utils:
    """这个类用于存储一系列的RAG工具类
    """
    def process_file(file_path):
        """_summary_

        Args:
            file_path (str): filepath
        """
        with open(file_path,encoding="utf-8") as f:
            text = f.read()
            # 将文本都变成一个一个的sentences，这里的实现的sentences是把文本以一
            sentences = text.split('\n')
            return text,sentences

    def generate_rag_prompt(data_point):
        """
        Args:
            data_point (str[]): 需要填充prompt的位置列表
        """
        return f"""Instruction:
            {data_point["instruction"]}
            ### Input:
            {data_point["input"]}
            ### Response:
        """
```

对文档进行Embedding:

Custom_Embedder.py:

```
import os
import argparse
import faiss
from transformers import AutoTokenizer, AutoModel, AutoModelForCausalLM
import torch
```

```
from tqdm import tqdm
class DocumentEmbedder:
    """这个类主要用于对需要知识库的文档进行embedding
    """
    def __init__(
            self,
            model_name='BAAI/bge-large-zh-v1.5',
            max_length=128,
            max_number_of_sentences=20
    ):
        """初始化embedder

        Args:
            model_name (str, optional):model_path. Defaults to 'BAAI/bge
            max_length (int, optional):maxlen for sentences. Defaults to
            max_number_of_sentences (int, optional): top k. Defaults to 2
        """
        #从模型仓库加载模型
        self.tokenizer = AutoTokenizer.from_pretrained(model_name)
        self.model = AutoModel.from_pretrained(model_name)
        #设置每个sentence的最长token数为多少
        self.max_length = max_length
        #设置需要考虑的最大sentence数量,K
        self.max_number_of_sentences = max_number_of_sentences

    def get_document_embeddings(self,sentences):
        """获取文本的embeddings

        Args:
            sentences (str[]):被切分的文本的str数组

        Returns:
            torch.tensor:返回的是整个文本的平均值
        """
        #只选取前K个sentences送入GPU
        sentences = sentences[:self.max_number_of_sentences]
        #先进行Tokenize
        encoded_input = self.tokenizer(sentences,
                                       padding=True,
                                       truncation=True,
                                       max_length=128,
                                       return_tensors="pt")
        #计算token的embeddings
        with torch.no_grad():
            model_ouput = self.model(**encoded_input)
```

```
        #文本的embedding 应该是里面所有sentences的平均
        #如果文本只有一个embeddding那么这个embedding就是这个文本的embedding
        return torch.mean(model_ouput.pooler_output,dim=0,keepdim=True)
```

针对给定的prompt进行回答的模型类：

model_generator.py

```
import os
import argparse
import faiss
from transformers import AutoTokenizer, AutoModel, AutoModelForCausalLM
import torch
from tqdm import tqdm

class GenerativeModel:
    """这是用于生成答案的模型
    """
    def __init__(self,
                 model_path="Qwen/Qwen1.5-7B",
                 max_input_length = 200,
                 max_generated_length = 200):
        """初始化LLM模型

        Args:
            model_path (str, optional): model_file_path. Defaults to "Qw
            max_input_length (int, optional): 最长输入token长度. Defaults
            max_generated_length (int, optional): 最长生成token长度. Defau
        """
        #加载LLM模型
        self.model = AutoModelForCausalLM.from_pretrained(
            model_path,
            device_map = "auto",
            torch_dtype = torch.float16,
        )

        self.tokenizer = AutoTokenizer.from_pretrained(
            model_path,
            padding_side = "left",
            add_eos_token = True,
            add_bos_token = True,
            use_fast = False,
        )
        self.tokenizer.pad_token = self.tokenizer.eos_token
        self.max_input_length = max_input_length
        self.max_generated_length = max_generated_length
```

```
        self.device = torch.device("cuda") if torch.cuda.is_available() e

    def answer_prompt(self,prompt):
        """生成prompt的答案

        Args:
            prompt (str): 提问LLM的语句

        Returns:
            _type_: _description_
        """
        messages = [
                {"role": "system", "content": "You are a helpful assistan
                {"role": "user", "content": prompt}
        ]
        text = self.tokenizer.apply_chat_template(
                messages,
                tokenize=False,
                add_generation_prompt=True
        )
        #先把prompt进行tokenize
        encoded_input = self.tokenizer([text],
                                       padding=True,
                                       truncation=True,
                                       max_length=self.max_input_length,
                                       return_tensors="pt")
        outputs = self.model.generate(input_ids=encoded_input['input_ids
                                      attention_mask = encoded_input['att
                                      max_new_tokens=self.max_generated_l
                                      do_sample = False)
        decoder_text = self.tokenizer.batch_decode(outputs,
                                                   skip_special_tokens=Tr
        return decoder_text
```

主方法：

```
import os
import argparse
import faiss
from transformers import AutoTokenizer, AutoModel, AutoModelForCausalLM
import torch
from tqdm import tqdm
from rag_utils import utils
from Custom_Embedder import DocumentEmbedder
```

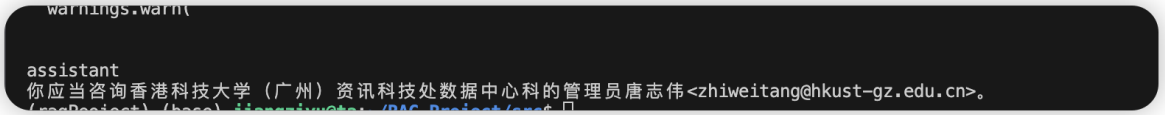
```
from model_generator import GenerativeModel
if __name__ == "__main__":
    parser = argparse.ArgumentParser()
    parser.add_argument("--documents_directory",
                        help="The directory that has the documents",
                        default='rag_documents'
                        )
    parser.add_argument("--embedding_model",
                        help="The HuggingFace path to the embedding model
                        default='BAAI/bge-large-zh-v1.5'
                        )
    parser.add_argument("--generative_model",
                        help="The HuggingFace path to the generative mode
                        default='Qwen/Qwen1.5-7B'
                        )
    parser.add_argument("--number_of_docs",
                        help="The number of relevant documents to use for
                        default=2
                        )
    args = parser.parse_args()
    '''
    把所有目录里的文件都chunk成sentences
    但是注意记录原始的文件地址
    '''
    print('Splitting documents into sentences')
    documents={}
    for idx,file in enumerate(tqdm(os.listdir(args.documents_directory)[
        current_filepath = os.path.join(args.documents_directory,file)
        #进行chunk
        text,sentences = utils.process_file(current_filepath)
        documents[idx] = {'file_path':file,
                          'sentences':sentences,
                          'document_text':text
                          }
    '''
    现在对所有chunk后的sentences进行embedding
    '''
    print('Getting document embeddings')
    #选择embedder并初始化
    document_embedder = DocumentEmbedder(model_name=args.embedding_model,
                                         max_length=128,
                                         max_number_of_sentences=20
                                         )
    embeddings = []
    #获得embedding
```

```
    for idx in tqdm(documents):
        #begin embedidng
        embeddings.append(document_embedder.get_document_embeddings(docu
    embeddings = torch.concat(embeddings,dim=0).data.cpu().numpy()
    embedding_dimensions = embeddings.shape[1]
    '''
    使用FAISS把所有embeddings建立索引
    '''
    faiss_index = faiss.IndexFlatIP(int(embedding_dimensions))
    faiss_index.add(embeddings)

    #输入你的问题
    question="请你告诉我，香港科技大学（广州） 一期HPC 平台资费问题我应该咨询哪个邮
    #对问题做embedding
    query_embedding = document_embedder.get_document_embeddings([questio
    distances,indices = faiss_index.search(query_embedding.data.cpu().nu
                                           k=int(args.number_of_docs)
                                           )
    '''
    使用k个最近的文件来为LLM提供上下文来生成answer
    '''
    context=''
    for idx in indices[0]:
        context += documents[idx]['document_text']
    rag_prompt = utils.generate_rag_prompt({'instruction':question,
                                            'input':context})
    '''
    使用LLM生成对应问题的答案，根据提供的context
    '''
    print('Generating answer...')
    generative_model = GenerativeModel(model_path=args.generative_model,
                                       max_input_length=2000,
                                       max_generated_length=2000
                                       )
    answer = generative_model.answer_prompt(rag_prompt)[0].split('### Re
    print(answer)
```

问题：

1. accelerate库安装了后，仍然报错，后发现是因为accelerate的版本太低了，pip install —upgrade后就好了
2. 使用了qwen1.5-7B模型之后，发现生成回答的速度很慢，而且回答的内容是原文档本身的内容，而不是问题的答案，所以重新选用了qwen1.5-4B-chat模型并且apply了chat_template,此时就能得到正确的答案了，现在遗留的问题是为什么使用7B非chat模型回答的内容是文档本身？



任务二：RAG版面分析模块

```
Group Notice
【非一般程序员训练营 第二季 —— RAG 潘多拉宝盒 任务二】
任务名称：RAG 版面分析模块
任务说明：
尝试对 ChatGLM评估挑战赛-金融赛道数据集 https://modelscope.cn/datasets/modelsc

本地知识文件获取涉及从多种数据源（如.txt、.pdf、.html、.doc、.xlsx、.png、.jpg、音

参考资料：
【版面分析—富文本txt读取】 https://articles.zsxq.com/id_x01sdalig305.html
【版面分析—PDF 解析神器 pdfplumber】 https://articles.zsxq.com/id_q02kt8h2c
【版面分析—PDF 解析神器 PyMuPDF】https://articles.zsxq.com/id_qmjw5wgbhs1s.
【版面分析—网页HTML解析 BeautifulSoup】https://articles.zsxq.com/id_ptqivfc
【版面分析—Docx 解析神器 python-docx】https://articles.zsxq.com/id_wshfpiwf
【版面分析—图片解析神器 OCR】 https://articles.zsxq.com/id_u8f1xl8v9hvd.html
【版面分析—OCR神器 PaddleOCR】https://articles.zsxq.com/id_zdut9rwjrh64.ht

任务截止时间：20240505（本周日）下午8点
任务打卡方式：本次活动采用【知识星球】打卡方式，即将个人学习文档（包含：项目运行效果、基
未打卡惩罚：送飞机票一张（退出本次学习活动）
本次活动最后将根据知
```

Step01: Download 数据集

```
# 方式1
git clone http://www.modelscope.cn/datasets/modelscope/chatglm_llm_fintec
```

下载了所有的pdf文件

但是文件众多，故只提取其中1000个pdf文件用作任务

统计当前目录下的文件数目的命令：

```
ls -l | grep "^-" | wc -l
```

随机选取1000个pdf到新文件夹下的命令

```
find /data/jiangziyu/data/chatglm_llm_fintech_raw_dataset/allpdf/ -name "
```

问题：发现没有安装过git lfs因此大文件没法下载下来，遂安装git lfs

Step02: 处理PDF文件，使用pdfplumber识别pdf文件的文本和表格信息，在远rag_utils.py文件中编写PDFProcessor类集成文本，表格提取，单个，多个pdf文件，图片提取以及表格文本提取转excel的方法以供pdf处理使用

```
import os
import argparse
import faiss
from transformers import AutoTokenizer, AutoModel, AutoModelForCausalLM
import torch
from tqdm import tqdm
import pdfplumber
from openpyxl import Workbook
from tqdm import tqdm
from concurrent.futures import ThreadPoolExecutor
class utils:
    """这个类用于存储一系列的RAG工具类
    """
    def process_file(file_path):
        """_summary_

        Args:
            file_path (str): filepath
        """
        with open(file_path,encoding="utf-8") as f:
            text = f.read()
            # 将文本都变成一个一个的sentences，这里的实现的sentences是把文本以一
            sentences = text.split('\n')
            return text,sentences

    def generate_rag_prompt(data_point):
        """
        Args:
            data_point (dict): 需要填充prompt的各个位置对应的dict
        """
        return f"""Instruction:
            {data_point["instruction"]}
            ### Input:
            {data_point["input"]}
            ### Response:
        """
class PDFProcessor:
    def __init__(self, input_folder, output_folder, max_workers=1):#切勿修
        self.input_folder = input_folder
```

```
        self.output_folder = output_folder
        self.max_workers = max_workers
        # 确保输出文件夹存在
        if not os.path.exists(output_folder):
            os.makedirs(output_folder)

    def process_pdf(self, filepath):
        """处理单个PDF文件，提取文本和表格。

        Args:
            filepath (str): 完整的PDF文件路径。
        """
        text_output_path = os.path.join(self.output_folder, os.path.basen
        table_output_path = os.path.join(self.output_folder, os.path.base

        # 打开PDF文件一次
        with pdfplumber.open(filepath) as pdf:
            text_content, table_content = self.extract_all_content(pdf)
            self.write_text(text_output_path, text_content)
            self.write_tables(table_output_path, table_content)

    def extract_all_content(self, pdf):
        """从PDF中提取所有文本和表格内容，存储在内存中，等待一次性写入."""
        text_content = []
        table_content = []
        for page in pdf.pages:
            # 提取文本
            textdata = page.extract_text()
            if textdata:
                text_content.append(textdata)
            # 提取表格
            for table in page.extract_tables():
                formatted_table = ["\t".join([str(cell) if cell else "" f
                table_content.append(formatted_table)
        return text_content, table_content

    def write_text(self, outputfile, text_content):
        """将所有提取的文本内容写入文件."""
        with open(outputfile, 'w', encoding='utf-8') as out_file:
            for text in text_content:
                out_file.write(text + "\n")

    def write_tables(self, outputfile, table_content):
        """将所有提取的表格内容写入文件."""
        with open(outputfile, 'w', encoding='utf-8') as out_file:
```

```
            for table in table_content:
                out_file.write("\n".join(table) + "\n\n")

    def create_excel(self, table_content, outputfile):
        """将提取的表格内容保存到Excel文件."""
        workbook = Workbook()
        sheet = workbook.active
        for table in table_content:
            for row in table:
                sheet.append(row.split('\t'))
        workbook.save(filename=outputfile)

    def process_all_pdfs(self):
        """使用多线程处理文件夹中的所有PDF文件，并显示进度条。"""
        pdf_files = [os.path.join(self.input_folder, f) for f in os.listd
        with ThreadPoolExecutor(max_workers=self.max_workers) as executor
            list(tqdm(executor.map(self.process_pdf, pdf_files), total=le
```

问题：由于发现处理速度太慢，找到原因是因为每识别读取一页就写入一页的内容到txt中去，有太多IO导致速度太慢，所以修改代码先识别读取PDF到内存中，再一次性写入到txt文件中减少io，同时使用了多线程以加速处理，后发现多线程存在问题，速度不如1个worker的处理速度，所以取消多线程的设置

随机选取了15个PDF文件来识别获得他们的文本txt和表格txt文件

效果如下

文本txt：

```
证券代码：000797 证券简称：中国武夷 公告编号：2020-044
债券代码：112301 债券简称：15中武债
债券代码：114495、114646 债券简称：19中武 R1、20中武R1
中国武夷实业股份有限公司
2019 年年度报告
2020 年 4 月
1
中国武夷实业股份有限公司2019年年度报告全文
第一节重要提示、目录和释义
公司董事会、监事会及董事、监事、高级管理人员保证年度报告内容的真
实、准确、完整，不存在虚假记载、误导性陈述或重大遗漏，并承担个别和连
带的法律责任。
公司负责人林增忠、主管会计工作负责人刘铭春及会计机构负责人(会计主
管人员)詹辉禄声明：保证年度报告中财务报告的真实、准确、完整。
所有董事均已出席了审议本报告的董事会会议。
公司需遵守《深圳证券交易所行业信息披露指引第 3 号—上市公司从事房
地产业务》的披露要求
公司已在本报告中详细描述未来将面临的主要风险和应对措施，详情请查
```

```
阅本报告“第四节经营情况讨论与分析”之“九、公司未来发展的展望”部分，
请投资者注意投资风险。
公司经本次董事会审议通过的利润分配预案为：以 2019 年 12 月 31 日总股
本 1,571,514,753 股为基数，向全体股东每 10 股派发现金红利 0.5 元（含税），
送红股 0 股（含税），不以公积金转增股本。
1
```

表格txt：

```
股票简称  中国武夷  股票代码  000797
股票上市证券交易所  深圳证券交易所
公司的中文名称      中国武夷实业股份有限公司
公司的中文简称      中国武夷
公司的外文名称（如有）  CHINA WUYI CO.,LTD.
公司的外文名称缩写（如有）  CHINA WUYI
公司的法定代表人    林增忠
注册地址  福建省福州市五四路89号置地广场4层
注册地址的邮政编码  350001
办公地址  福建省福州市五四路89号置地广场4层
办公地址的邮政编码  350001
公司网址  http://www.chinawuyi.com.cn/
电子信箱  gzb@chinawuyi.com
```

观察发现识别后的文本和表格txt都存在非常严重的形式，格式缺失，虽然保有大部分文字内容，但是格式已不可辨认，表格资料这一点最为明显

准备将这个资料库用于形成RAG模型的知识库的时候发现，不能对整个文本进行embedding，因为文本太长了，超过了模型的token限制，所以这一点如何处理是个难点