数据采集

* 1. **数据来源**

以较受欢迎的百度旗下的汽车信息与服务平台有驾（yoojia）作为获取数据的网页，通过爬虫技术获取网站上的秦PLUS、宋PLUS两款汽车的评论，包含用户的id、汽车分数、成交价格、成交时间、交易地点和用户评论等共5221条数据。

**1.2爬虫技术实现**

网络爬虫（Web Crawler）是一种自动化程序，用于遍历互联网并收集信息。其工作方法从种子URL开始，这些URL是爬虫的起点。爬虫访问这些种子URL，发送HTTP请求并接收响应，以下载网页内容。接着，爬虫解析下载的网页，提取其中的链接、文本、图像等信息，这通常使用HTML解析库如BeautifulSoup或lxml。提取的信息存储在数据库或文件中，以便后续分析和处理。在解析网页内容时，爬虫还会提取新的链接，并将这些链接添加到待抓取的URL队列中。这个过程不断重复，直到满足预定的停止条件，如达到抓取深度或数量限制。爬虫系统的基本流程包括3个部分。第一步，通过网络协议向目标网站发送请求。网络信息平台多使用HTTP和HTTPS。第二步，获取信息。第三步存储数据到json文件中。

1.2.1 代码实现

导入库

import **requests**

import **json**

import **time**

import **re**

导入代码所需要的python库，其中requests库是请求库，用于发送HTTP请求，json库用于处理json数据，time库用于控制请求之间的延迟。

常量定义

MOBILE\_TOKEN = '1\_2298dc2ac944caad6d611cdbbcb9710e'

DESKTOP\_TOKEN = '1\_526c1239fc0b0512a2bd13ac6b962f5f'

DATA\_PRODUCER = 2

DELAY = 0.1  *# 请求之间的延迟，单位为秒*

DESKTOP\_TAGS = [

    'all', 'space', 'fuel', 'cruising', 'workmanship,internal,external',

    'powersteering,power,control', 'comfort', 'cp', 'select'

]

MOBILE\_TAGS = [

    'all', 'space', 'consumption', 'cruising', 'workmanship',

    'powersteering', 'select\_process', 'cp'

]

SERIES\_IDS = ['1005964']

HEADERS = {

    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/91.0.4472.124 Safari/537.36',

    'Accept': 'application/json, text/javascript, \*/\*; q=0.01',

    'Accept-Language': 'en-US,en;q=0.9',

    'Connection': 'keep-alive',

    'Referer': 'https://www.yoojia.com',

}

MOBILE\_TOKEN与DESKTOP\_TOKEN用于身份验证，以便从有驾网站的API接口获取数据。DATA\_PRODUCER是数据来源标识用于标识数据来源的类型，区分数据的不同的生产者。此处设为2确保API请求的是用户评论而非其他数据。DELEY用于控制请求之间的延迟，提高请求的成功率，避免频繁请求对服务器造成压力，导致触发反爬虫机制。DESKTOP\_TAGS与MOBILE\_TAGS是标签，用于分类不同类型的评论。SERIES\_ID用于获取特定汽车系列的ID，用于获取秦PLUS与宋PLUS的数据。HEADERS 是一个包含HTTP请求头信息的字典，用于模拟浏览器访问API接口，让爬虫伪装成正常的浏览器访问，从而绕过一些简单的反爬虫机制。

函数定义

def **fetch\_comments**(page\_number, series\_id, tag, sort=0, max\_retries=5, delay=1):

    url = f'https://www.yoojia.com/gateway/koubei-api/koubeis/format?token={DESKTOP\_TOKEN}&pn={page\_number}&rn=10&tab={tag}&data\_producer={DATA\_PRODUCER}&series\_id={series\_id}&monitor=USER\_COMMENT\_pc&sort={sort}'

    for attempt in **range**(max\_retries):

        try:

            response = **requests**.**get**(url, headers=HEADERS)

            response.**raise\_for\_status**()  *# 检查HTTP请求是否成功*

            return response.**json**()

        except **requests**.**exceptions**.**RequestException** as e:

**print**(f"Error fetching page {page\_number}, series\_id {series\_id}, tag {tag} on attempt {attempt + 1}. Error: {e}")

**time**.**sleep**(delay)

**print**(f"Failed to fetch page {page\_number}, series\_id {series\_id}, tag {tag} after {max\_retries} attempts.")

    return None

该函数用于获取特定页面的评论数据，构建请求URL，包含页码、系列ID、标签和排序参数。尝试最多 max\_retries 次获取数据，如果成功则将HTTP响应的内容解析为JSON格式的数据并返回，否则打印错误并等待 delay 秒后重试。

def **parse\_comments**(data):

    comments = data.get('data', {}).get('koubei\_list', [])

    parsed\_comments = []

    for comment in comments:

        parsed\_comments.**append**({

            'id': comment.get('id'),

            'car\_price': comment.get('car\_purchase\_price'),

            'car\_buytime': comment.get('purchase\_time'),

            'car\_buylocation': comment.get('car\_purchase\_location'),

            'car\_name': comment.get('car\_info', {}).get('car\_model\_name'),

            'comment': **extract\_comment\_content**(comment.get('content', []))

        })

    return parsed\_comments

该函数用于解析评论数据，将其转换为所需的格式，提取评论的ID、购车价格、购车地点、购车时间、车型以及评论内容。

def **extract\_comment\_content**(content):

    full\_text = []

    for item in content:

        full\_text.**append**(item.get('text', ''))

    return ' '.**join**(full\_text)

该函数用于提取评论内容，并将其拼接为一个字符串

主函数

def **main**(series\_ids, tags):

    for series\_id in series\_ids:

        all\_comments = []

        for tag in tags:

            sort\_range = **range**(2) if tag == 'all' else **range**(1)

            for sort in sort\_range:

                page\_number = 1

                while True:

**print**(f'Fetching page {page\_number}/76 for series\_id {series\_id} with tag {tag}...')

                    data = **fetch\_comments**(page\_number, series\_id, tag,sort=sort)

                    if data is None:

                        break

                    comments = **parse\_comments**(data)

                    if not comments:

**print**(f'No more comments found for series\_id {series\_id} at page {page\_number} with tag {tag}.')

                        break

                    all\_comments.**extend**(comments)

                    page\_number += 1

*# 判断是否还有更多评论*

                    if data.get('data', {}).get('koubei\_list') == []:

**print**(f'No more comments available for series\_id {series\_id} with tag {tag}.')

                        break

**time**.**sleep**(DELAY)  *# 添加延迟*

        file\_name = f'desktop\_comments\_{series\_id}.json'

        with **open**(file\_name, 'w', encoding='utf-8') as f:

**json**.**dump**(all\_comments, f, ensure\_ascii=False, indent=4)

**print**(f'Comments for series\_id {series\_id} saved to {file\_name}.')

该函数用于遍历所有系列的ID和标签，同时输出是否成功获取到了评论，并将成功获取到的评论数据解析，保存到当地的json文件中。

主程序

if \_\_name\_\_ == "\_\_main\_\_":

    series\_ids = SERIES\_IDS

    tags=DESKTOP\_TAGS

**main**(series\_ids,tags)

初始化参数。

import **json**

import **hashlib**

import **os**

def **hash\_comment**(comment):

    """

    对评论内容进行哈希处理

    """

    return **hashlib**.**md5**(comment.encode('utf-8')).**hexdigest**()

def **remove\_duplicates**(input\_file, output\_file, backup\_folder):

    """

    从输入文件中读取评论，去重后保存到输出文件

    """

    with **open**(input\_file, 'r', encoding='utf-8') as infile:

        comments = **json**.**load**(infile)

    unique\_comments = []

    seen\_hashes = {}

    duplicate\_count = 0

    diff\_id\_count = 0

    if not **os**.path.**exists**(backup\_folder):

**os**.**makedirs**(backup\_folder)

    for comment in comments:

        comment\_text = comment['comment']

        comment\_hash = **hash\_comment**(comment\_text)

        if comment\_hash in seen\_hashes:

            duplicate\_count += 1

            if seen\_hashes[comment\_hash] != comment['id']:

                diff\_id\_count += 1

                backup\_file = **os**.path.**join**(backup\_folder, f"{seen\_hashes[comment\_hash]}\_{comment['id']}.json")

                with **open**(backup\_file, 'w', encoding='utf-8') as backup\_f:

**json**.**dump**([seen\_hashes[comment\_hash], comment], backup\_f, ensure\_ascii=False, indent=4)

        else:

            seen\_hashes[comment\_hash] = comment['id']

            unique\_comments.**append**(comment)

    with **open**(output\_file, 'w', encoding='utf-8') as outfile:

**json**.**dump**(unique\_comments, outfile, ensure\_ascii=False, indent=4)

    total\_comments = **len**(comments)

    unique\_comments\_count = **len**(unique\_comments)

    backup\_files\_count = **len**(**os**.**listdir**(backup\_folder))

**print**(f"去重完成，去重后的评论已保存到 {output\_file}")

**print**(f"总评论数: {total\_comments}")

**print**(f"唯一评论数: {unique\_comments\_count}")

**print**(f"重复评论数: {duplicate\_count}")

**print**(f"重复但ID不同的评论数: {diff\_id\_count}")

**print**(f"备份文件数: {backup\_files\_count}")

if \_\_name\_\_ == "\_\_main\_\_":

    input\_files = ['desktop\_comments\_1005761.json', 'desktop\_comments\_1005964.json']

    output\_files = ['desktop\_宋PLUS新能源.json', 'desktop\_秦PLUS.json']

    backup\_folder = 'backup\_duplicates'

    for input\_file, output\_file in **zip**(input\_files, output\_files):

**remove\_duplicates**(input\_file, output\_file, backup\_folder)

同时，我们考虑到可能会有重复的数据出现，所以我们需要以上代码来处理和去重数据。以上代码从输入文件中读取评论数据，利用哈希函数对评论内容进行哈希处理，从而检测重复评论。对于重复评论，程序会根据哈希值进行比较，如果发现不同ID的重复评论，则将这些评论备份到指定的备份文件夹中。同时，程序会记录总评论数、唯一评论数、重复评论数以及不同ID但内容重复的评论数，并将去重后的评论保存到输出文件中。

首先，定义函数，接受评论内容并返回其MD5哈希值，用于唯一标识评论内容。接着主函数读取输入文件中的评论数据，并初始化一个空列表用于存储唯一评论，以及一个字典用于跟踪已经遇到过的评论的哈希值。然后遍历所有评论，获取哈希值并检测其是否在字典中。如果相同则根据ID判断是否相同。如果哈希值不存在于字典中，则将该评论视为唯一评论并添加到唯一评论列表中，同时将哈希值及其对应的评论ID记录到字典中。

1.2.2 实例

上述两张图是通过修改SERIES\_ID获取到的比亚迪秦PLUS和宋PLUS两款车型的示例，可以看到，爬虫成功的将用户评论的相关数据存储到了本地的json文件中

1.3基于jieba的中文分词

分词是自然语言处理的一个基本步骤，其目的是将一段长文本通过分词工具jieba、NLTK、spaCy等拆分成基本单元（词与子词），使得其可以作为模型输入，便于后续的工作开展。鉴于爬虫收集的是中文文本，我们采用适用中文文本的分词工具jieba来进行分词操作。

1.3.1代码实现

*# -\*- coding: utf-8 -\*-*

import **json**

import **pandas** as **pd**

import **re**

import **jieba**

import **logging**

import **os**

from **torchtext**.**data** import **Field**, **TabularDataset**, **Iterator**

*# 设置 jieba 日志等级*

**jieba**.**setLogLevel**(**logging**.INFO)

*# 定义正则表达式，只保留中文字符*

regex = **re**.**compile**(r'[^\u4e00-\u9fa5]')

*# 分词函数*

def **word\_cut**(text):

    text = regex.**sub**(' ', text)

    words = [word for word in **jieba**.cut(text) if word.**strip**() and word not in stopwords]

**print**(f"原文本: {text}")

**print**(f"分词后: {words}")

    return words

*# 数据集加载函数*

def **get\_dataset**(path, text\_field, label\_field):

    text\_field.tokenize = **word\_cut**

    train, valid, test = **TabularDataset**.**splits**(

        path=path, format='tsv', skip\_header=True,

        train='train.tsv', validation='valid.tsv', test='test.tsv',

        fields=[

            ('id', None),  *# 根据数据结构调整字段名称*

            ('car\_name', None),  *# 根据数据结构调整字段名称*

            ('car\_score', None),  *# 根据数据结构调整字段名称*

            ('car\_price', None),  *# 根据数据结构调整字段名称*

            ('car\_buytime', None),  *# 根据数据结构调整字段名称*

            ('car\_buylocation', text\_field),  *# 根据数据结构调整字段名称*

            ('comment', text\_field),  *# 根据数据结构调整字段名称*

        ]

    )

    return train, valid, test

*# 读取 JSON 文件*

json\_path = 'C:\\Users\\方俊皓\\Desktop\\spider\\秦PLUS.json'

if not **os**.path.**exists**(json\_path):

**print**(f"路径不存在: {json\_path}")

else:

    with **open**(json\_path, 'r', encoding='utf-8') as file:

        data\_content = **json**.**load**(file)

*# 创建 DataFrame 并保存为 TSV 文件*

    df = **pd**.**DataFrame**(data\_content)

    tsv\_dir = 'C:\\Users\\方俊皓\\Desktop\\spider'

    tsv\_path = **os**.path.**join**(tsv\_dir, 'train.tsv')

    df.**to\_csv**(tsv\_path, sep='\t', index=False)

*# 创建验证集和测试集*

    valid\_data = df.**sample**(frac=0.1, random\_state=42)  *# 10%的数据作为验证集*

    remaining\_data = df.**drop**(valid\_data.index)

    test\_data = remaining\_data.**sample**(frac=0.1, random\_state=42)  *# 10%的剩余数据作为测试集*

    train\_data = remaining\_data.**drop**(test\_data.index)

    valid\_path = **os**.path.**join**(tsv\_dir, 'valid.tsv')

    test\_path = **os**.path.**join**(tsv\_dir, 'test.tsv')

    train\_path = **os**.path.**join**(tsv\_dir, 'train.tsv')

    train\_data.**to\_csv**(train\_path, sep='\t', index=False)

    valid\_data.**to\_csv**(valid\_path, sep='\t', index=False)

    test\_data.**to\_csv**(test\_path, sep='\t', index=False)

*# 定义字段*

    TEXT = **Field**(sequential=True, tokenize=**word\_cut**)

    LABEL = **Field**(sequential=False, use\_vocab=False)

*# 加载数据集*

    train\_data, valid\_data, test\_data = **get\_dataset**(tsv\_dir, TEXT, LABEL)

*# 构建词汇表*

    TEXT.**build\_vocab**(train\_data, valid\_data, test\_data)

*# 创建迭代器*

train\_iter, valid\_iter, test\_iter = **Iterator**.**splits**(

    (train\_data, valid\_data, test\_data),

    batch\_sizes=(32, 256, 256),

    sort\_key=lambda x: **len**(x.comment),

    sort\_within\_batch=False,

    repeat=False

)

*# 打印一个批次的数据*

for batch in train\_iter:

**print**("一个批次的数据:")

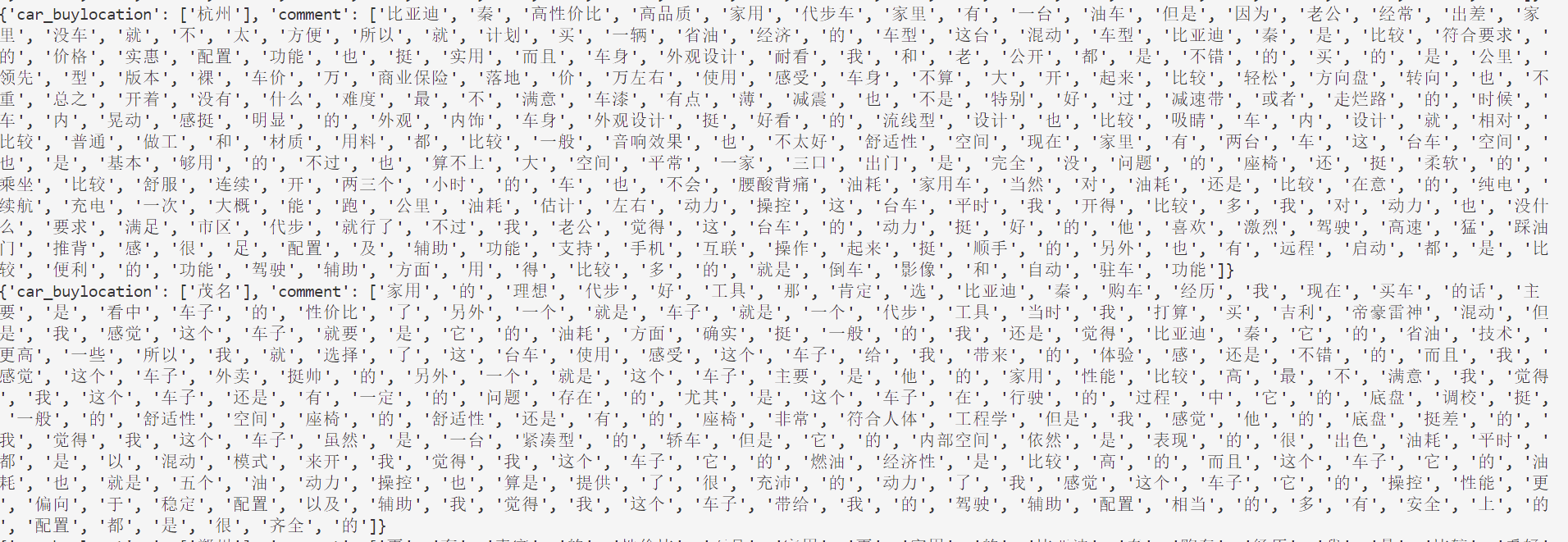
**print**(batch.comment)

    break

在上述代码中，我们先将jieba的日志等级设置，减少不必要的日志输出，通过正则表达式只保留爬取数据中的中文，并将剩余的字符使用word\_cut全部用空格替代以达到过滤文本的目的，然后使用jiaba进行分词操作。将get\_dataset函数用于加载数据集。最后通过调用定义的函数来实现分词的操作。

1.3.2 输出样例

以下是通过jieba分词工具以及python代码实现的分词操作，为了方便比较，我们同时输出了原文本以及分词后的效果。可以看到，以上代码成功达到了分词的目的。

1.3.3 分词的意义

分词将原始的文本数据处理成计算机能够理解和处理的形式。这是自然语言处理（NLP）中的一个关键步骤，因为自然语言文本通常是一个连续的字符串，对于计算机而言，这样的形式不便于直接进行分析和处理。分词通过将文本切分成独立的词语或子词，使得文本变得结构化，更加易于进行后续的处理和分析。

分词可以显著提高文本处理的准确性和效率。将文本切分成更小的单元后，统计、查询和分析变得更加容易。这种处理方式还便于将文本转化成向量，例如词袋模型（Bag-of-Words）和词嵌入（Word Embedding）等形式，从而捕捉文本的语义信息。这种语义信息的捕捉对于提高机器学习和深度学习模型的性能非常重要，因为它能够更好地理解和解释文本内容。

在情感分析等应用中，分词尤为重要。通过分词，可以识别出文本中的关键情感信息词语，这些词语对于判断文本的情感倾向至关重要。例如，分词可以将句子“我非常喜欢这款产品”分解为“我”、“非常”、“喜欢”、“这款”、“产品”，从而识别出“喜欢”这个表示积极情感的词语。这种识别可以帮助情感分析模型更加准确地判断文本的情感倾向，无论是正面、负面还是中性情感。

在数据清洗与预处理阶段，分词也发挥了重要作用。通过分词，可以去除停用词、标点符号等无关信息，使得数据更加干净整洁。停用词是指在文本处理中被过滤掉的常见词语，例如“的”、“了”、“是”等，它们通常对文本的主要内容没有贡献。去除这些停用词和无关信息后，数据变得更加简洁，模型可以更加专注于重要的信息，从而提高模型的性能。

此外，分词还可以减少文本的长度，节省计算资源，提高处理速度。在处理长文本时，分词可以将文本拆分成更小的段落或句子，便于并行处理。这种方式不仅提高了处理速度，还使得计算资源的使用更加高效。