Web LAb1

Part 1

Section 1 分词

我是用了两种现成的分词工具，并自己加入了一些分词优化以使得分词的结果更符合后续实验要求。两种的现成的分词工具分别为jieba分词和pkuseg分词。完成代码时我并没有特别考虑到拓展性，所以我使用的时面向过程编程，分别对book和movie使用jieba分词和pkuseeg分词，这里我们使用jieba分词对book的tag的分词为例子。

```
T1 = time.time()
for i,member in enumerate(book_reader):
    if(i==0): continue
    is_pair = 0
    str = ""

    book_output_row = ""
    book_output_row += member[0] + "," + '"' + "{"

    words = []
    # print(member[1])
    for ch in member[1]:
        if((ch == "," or ch == '}')and is_pair == 1):
            is_pair = 0
            words.append(str[:-1])
            str = ""
            continue
        if(is_pair == 1):
            str += ch
        if(ch == "'" and is_pair == 0): is_pair = 1
    token = []
    for word in words:
        tokens = ','.join(jieba.cut(word, HMM=True))
        for token in tokens.split(','):
            f = 0
            for bad in bad_list:
                if(token == bad):
                    f = 1
                    break
            if(f == 0):
                book_output_row += "'" + token + "'" + ","
    book_output_row += "}" + '"' + "\n"
    book_content.append(book_output_row)

T2 = time.time()
print("jieba book time: ", T2-T1)

with open(book_output_path, 'w') as f:
    f.write(book_column + "\n")
    f.writelines(book_content)
```

具体的分词方式我们使用调包完成，代码的中间部分是对源csv文件的读取。原来的Tags使用的是一个由单引号、逗号、Tag等组成的字符串，这里把其中真正的Tag进行抽取并进行分词，最终以相同的格式重新储存到文件里面。其中值得一提的是，我们根据前期工作分词的结果，筛选出了我们不想要的一些代表不了信息的词汇汇总为bad_list，在提取时碰到这些词汇不会加入到分词表中。

接下来分析Jieba分词和PKUSeg分词

Jieba 分词基于 词典+HMM（隐马尔科夫模型） 的统计方法，使用双向最大匹配（Bi-MM）来匹配词典中的词语。HMM模型用于处理未登录词（未出现在词典中的词），通过字词的状态转移概率来预测分词边界。(本次实验中我们并未使用jieba分词的paddle模式，即使用paddle深度学习框架，我们也没能使用jieba分词的自定义字典)

PKUSeg 分词基于条件随机场（CRF）或深度学习模型的序列标注方法。使用大规模标注语料训练词边

界的概率分布，通过全局最优来预测分词结果。

```
(web) sakiko@LAPTOP-9TSVN48O:~/Web/lab/web-lab1/src_lwb$ python cutter_jieba.py
Building prefix dict from the default dictionary ...
Loading model from cache /tmp/jieba.cache
Loading model cost 0.494 seconds.
Prefix dict has been built successfully.
jieba book time:  4.415745973587036
jieba movie time:  16.06471824645996
(web) sakiko@LAPTOP-9TSVN48O:~/Web/lab/web-lab1/src_lwb$ python cutter_pkuseg.py
pkuseg book time:  10.418039798736572
pkuseg movie time:  46.36260175704956
```

1. 性能对比
   准确性: PKUSeg 在复杂场景和领域适配上表现优异，因为其模型能捕捉上下文和细粒度语义。
   速度: Jieba 因为算法简单，分词速度更快，适合大规模实时处理任务。
   未登录词处理: Jieba 依赖 HMM，可能效果有限；而 PKUSeg 的深度学习方法能够更好地识别新词。
2. 性能差异原因
   算法复杂度: Jieba 是启发式规则与简单统计方法，性能上更轻量；PKUSeg 基于训练数据和深度学习模型，对算力要求更高。
   数据依赖: PKUSeg 需要高质量标注数据进行训练，适应性强；而 Jieba 的性能主要受词典的质量限制。
   上下文建模能力: PKUSeg 的深度学习模型能有效捕捉上下文关系；Jieba 的 HMM 模型对于长距离依赖能力较弱。

Section 2

通过上一阶段生成的分词结果，生成倒排索引表，处理过程中采用{(word，[id1, id2...])...}的格式存储。

首先通过基础方式存储，即对于词典，采取[88byte词项，4byte频率，4byte倒排索引指针]方式顺序存储（发现上述分词结果中最长的词项为85bytes，所以采用88bytes存储，空位使用b'\x00'填充）。

对于倒排索引文件，采用4字节每文档id的方法顺序存储。

（上述生成倒排索引表和词典文件的方法在"web-lab1\src_yzy\inverted_index_gen.py"中，采用基本顺序存储的读取方法测试（从二进制文件还原倒排索引表）包含在"web-lab1\src_yzy\test_basic.py"中）

具体建立倒排索引表的过程如下(in web-lab1\src_yzy\inverted_index_gen.py)：

```python
def insert_line(i, textline):
    global inverted_index
    if i == 0:
        return

    index = int(textline[0])    #book id
    words = []                  #tags
    is_pair = 0
    str = ""

    for ch in textline[1]:
        if((ch == "," or ch == '}')and is_pair == 1):
            is_pair = 0
            words.append(str[:-1])
            str = ""
            continue
        if(is_pair == 1):
            str += ch
        if(ch == "'" and is_pair == 0): is_pair = 1
    for word in words:
        set = inverted_index.setdefault(word, [])   #不使用三步走，直接使用字典+列表
```

```
        set.append(index)
```

用于处理一行对应的记录。

基础的顺序存储实现如下(in web-lab1\src_yzy\inverted_index_gen.py)(去除了编写时的一些测试项):

```
def write_index_normal(file_path):  #需要分开存储词典和倒排表项
    global inverted_index
    #测试得到最长词项为85字节，于是取88字节存储词项
    dict_pt = 0
    index_pt = 0
    file1 = open(file_path + "_basic_store_dict.binery", "wb")
    file2 = open(file_path + "_basic_store_index.binery", "wb")
    for index,(key, value) in enumerate(inverted_index):
        word = key.encode() + (88 - len(key.encode())) * b'\x00'
        file1.write(word)
        dict_pt += 88
        file1.seek(dict_pt)
        file1.write(len(value).to_bytes(4))
        dict_pt += 4
        file1.seek(dict_pt)
        file1.write(index_pt.to_bytes(4))
        dict_pt += 4
        file1.seek(dict_pt)
        for id in value:
            file2.write(id.to_bytes(4))
            index_pt += 4
            file2.seek(index_pt)
    return
```

与之对应的，从二进制文件中还原的过程如下(in web-lab1\src_yzy\test_basic.py):

```
input = ["../data/index/selected_book_top_1200_data_tag_tokenized_jieba",
"../data/index/selected_book_top_1200_data_tag_tokenized_pkuseg",
"../data/index/selected_movie_top_1200_data_tag_tokenized_jieba",
"../data/index/selected_movie_top_1200_data_tag_tokenized_pkuseg"]
output =
["../data/index/testread/selected_book_top_1200_data_tag_tokenized_jieba",
"../data/index/testread/selected_book_top_1200_data_tag_tokenized_pkuseg",
"../data/index/testread/selected_movie_top_1200_data_tag_tokenized_jieba",
"../data/index/testread/selected_movie_top_1200_data_tag_tokenized_pkuseg"]

def process(i):
    file_dict = open(input[i] + "_basic_store_dict.binery", "rb")
    file_index = open(input[i] + "_basic_store_index.binery", "rb")
    file_dict_out = open(output[i] + "_dict.txt", "w", encoding="utf-8")
    file_index_out = open(output[i] + "_index.txt", "w", encoding="utf-8")

    dict_data = file_dict.read()
    index_data = file_index.read()

    dict_ptr = 0
    index_ptr = 0
    print(len(dict_data))
    while (dict_ptr < len(dict_data)):
        word = (dict_data[dict_ptr : dict_ptr +
88].rstrip(b'\x00')).decode('utf-8')
```

```
        freq = int.from_bytes(dict_data[dict_ptr + 88 : dict_ptr + 92])
        pindex = int.from_bytes(dict_data[dict_ptr + 92 : dict_ptr + 96])
        dict_ptr += 96
        print(word, freq, pindex, file=file_dict_out)
        index = []
        for i in range(0, freq):
            index.append(int.from_bytes(index_data[index_ptr : index_ptr + 4]))
            index_ptr += 4
        print(index, file=file_index_out)
```

实际上除了个别长词之外，一般的词项远远达不到88字节，我们考虑对词典文件进行压缩。除去基本存储方法以以外，试图采用单一字符串词典以及按块存储方法，并比较这三种方法的空间占用以及检索效率。

单一字符串方法采用：开头存储二进制长字符串字节数，然后存放对应大小的字符串，接下来每12字节中依次存放{4字节字符串指针，4字节词项频率，4字节倒排索引表指针}

按块存储采用：开头存放字符串方法相同，然后每5项存放一次字符串指针，接着对于每一项的9字节：{1字节词项字符串长度，4字节频率，4字节倒排索引表指针}

相关的存放代码以及读取方式在如下文件中实现：

"web-lab1\src_yzy\word_compress_store_single.py"

"web-lab1\src_yzy\word_compress_store_block.py"

"web-lab1\src_yzy\test_single.py"

"web-lab1\src_yzy\test_block.py"

关于空间占用，结果如下：

（首先在第一阶段中，我们对于book和movie使用两种工具进行了分词，以下是共四组倒排索引表使用三种词典压缩方法后的空间占用对比，相应的倒排索引表文件大小也包括在内，由于对倒排索引表没有压缩因此大小不存在变动）

	book_jieba	book_pkuseg	movie_jieba	movie_pkuseg
顺序存储	1836KB	1844KB	4958KB	5780KB
单一字符串	355KB	363KB	984KB	1230KB
按块存储	313KB	320KB	870KB	1098KB
倒排索引表	1180KB	1184KB	4817KB	4512KB

从上面的结果可以看出，从简单的顺序存储（每一项使用88字节保证存储超长词汇）到单一字符串的改进大大缩小了词典文件的存储，可见大量的短字符串是达不到88字节的长度的。

从单一字符串到按块索引的改进进一步缩小了词典文件的大小（在这里设置每5个块放置一次字符串指针），理论上除了开始的长字符串可以每60字节节省12字节，实际的表现也非常符合预期结果。

Section 3

Section 3

Part 2

Section 1

我们使用了助教提供的数据集进行数据划分。

由于数据集中的数据较为稠密，因此没有进行删减与更换。

通过修改助教提供的数据划分阶段代码，我们最终决定以 训练集：测试集 = 7 : 3 的比例划分数据。

即：train_data, test_data = train_test_split(loaded_data, test_size=0.3, random_state=42)

Section 2 基本协同过滤—基于项目

我们采取基于项目的基本协同过滤方式，预测用户对项目的评分。

选择基于项目而不是基于用户的推荐，是考虑到各类用户的偏好不同，而项目的属性较为单一，评分的标准相较于各人的喜好也更为一般化。基于项目的推荐相较于基于用户的推荐一般会更好。

基于项目推荐的计算公式如下：
$$r*{ix} = \frac{\Sigma_j \epsilon N(i;x) s*{ij}·r*{jx}}{\Sigma S*{ij}}$$

其中，$$s_{ij}$$代表i、j项目的相似程度，在此我们使用简单的$$Pearson$$相关系数表示，使用基于同一用户进行的评分进行衡量。具体来说，就是对于i、j项目，构造两个向量，每一维代表某一个用户对该项目的评分，而两个向量的同一维来自于同一个用户的评价。

$$r_{jx}$$则代表x用户对j项目的评分

Section 3 协同过滤进阶

Section 4 评价预测结果-基于 NDCG

最终我们使用 NDCG 来评价本次实验评分预测中的预测效果。

NDCG 的计算公式为：

$$ND$$

```
NDCG@2000: 0.9728067955683188
```

```
NDCG@2000: 0.9754043025001476
```