信息检索与数据挖掘实验一

数据 18 刁龙飞 201800130083

2020 年 10 月 5 日

1 环境配置

1.1 软件环境

Jupyte Notebook(Python 3.7.0)

2 前期准备

2.1 读入数据

用 Python 中的列表结构来存储读入的数据。对于 tweets 文件，一次读入一行。由于源文件是用字典来表示，对于每一条 tweet，将键'tweetId'和'text' 对应的值存入列表中。最终返回一个二维列表。

2.2 数据预处理

数据预处理分为以下步骤：大小写转换、特殊符号替换、去除停用词、词干还原。

```
In [6]: tweets_open = open('tweets.txt','r')
        data_ori = ReadData(tweets_open)
        tweets_open.close()
        print(data_ori[:10]) #读取的原始数据
```

```
[['2896579281289216O', 'House may kill Arizona-style immigration bill, Rep. Rick Rand says: The House is unlikely to pass the "Ari... htt
p://tinyurl.com/4jrjcdz.'], ['2896709587828736O', "Mourners recall Sarge Shriver's charity, idealism \n    (AP): AP - R. Sargent Shriver was
always an optimist, pio... http://bit.ly/gqMcdG"], ['2896767207499366 4', 'Bass Fishing Techniques: 2 Fantastic Tips To Improve Your Casting
Skills'], ['2896791441768857 6', '#Financial Aid | Proper Method Of Getting Financial Aid For Education http://ping.fm/BK0R3 #applying-for-fi
nancial-aid financial-aid-essay #'], ['2896847917653196 9', "Supreme Court: NASA's intrusive background checks OK http://bit.ly/h2jgy9"], ['2
8968581949558787', 'The McDonalds music to fireworks is an all time low.'], ['2896942205607116 9', "@alyce Very sweet and quiet, if not polis
hed - Bono & Hansard at Sgt Shriver's funeral 2day: http://youtu.be/Bf14XBbcVZg (when was ...cont'd"], ['2897174996189184O', "So, Avon&Somer
set Police have charged Vincent Tabak with the murder of Jo Yeates. I really hope they're right, otherwise his life is ruined."], ['28973080
491589632', 'Hawaii Gov Waffles on Obama's Birth Certificate - Patriot Update http://t.co/lUxYaOr via @AddThis'], ['2897486203899494 5', 'I
ve never retweeted myself but wanted to pass on to @atu2 RT @tommymcgregor: I Want Bono To Sing At My Funeral! http://bit.ly/iOKdEn']]
```

图 1: 读入数据

```
In [6]: id_list_file = open('all_text.txt','w',encoding='utf-8')
        id_list_file.write(str(data_ori))
        id_list_file.close()
        data = DataPreprocess(data_ori)
        print(data[:10]) #预处理后的数据
```

```
[['2896579281289216O', 'hous', 'may', 'kill', 'arizona', 'style', 'immigr', 'bill', 'rep', 'rick', 'rand', 'say', 'hous', 'unlik', 'pass',
'the', 'ari', 'http://tinyurlcom/4jrjcdz'], ['2896709587828736O', 'mourner', 'recal', 'sarg', "shriver'", 'chariti', 'ideal', 'ap', 'ap',
'r', 'sargent', 'shriver', 'alway', 'optimist', 'pio', 'http://bit.ly/gqmcdg'], ['2896767207499366O', 'bass', 'fish', 'techniqu', '2', 'fant
ast', 'tip', 'improv', 'your', 'cast', 'skill'], ['2896791441768857O', 'financi', 'aid', 'proper', 'method', 'get', 'financi', 'aid', 'edu
c', 'http://ping.fm/bkOr3', 'appli', 'for', 'financi', 'aid', 'financi', 'aid', 'essay'], ['2896847917653196O', 'suprem', 'court', "nasa'",
'intrus', 'background', 'check', 'ok', 'http://bit.ly/h2jgy9'], ['2896858194955878O', 'mcdonald', 'music', 'firework', 'all', 'time', 'lo
w'], ['2896942205607116O', '@alyc', 'veri', 'sweet', 'quiet', 'if', 'not', 'polish', 'bono', 'hansard', 'sgt', "shriver'", 'funer', '2day',
'http://youtu.be/bf14xbbcvzg', 'when', "cont'd"], ['2897174996189184O', 'so', 'avon', 'somerset', 'polic', 'have', 'charg', 'vincent', 'taba
k', 'murder', 'jo', 'yeat', 'i', 'realli', 'hope', "they'r", 'right', 'otherwis', 'hi', 'life', 'ruin'], ['2897308049158963O', 'hawaii', 'go
v', 'waffl', 'obama', '', 'birth', 'certif', 'patriot', 'updat', 'http://t.co/luxya0r', 'via', '@addthi'], ['2897486203899494O', 'ive', 'neve
r', 'retweet', 'myself', 'but', 'want', 'pass', 'to', '@atu2', 'rt', '@tommymcgregor', 'i', 'want', 'bono', 'to', 'sing', 'my', 'funer', 'ht
tp://bit.ly/iOkden']]
```

图 2: 预处理后的数据

2.2.1 大小写转换

使用 Python 字符串的.lower() 方法，将所有'text' 转换成小写。

2.2.2 特殊符号替换

此处定义的特殊符号替换规则较为复杂。一方面，我们希望尽可能去除 tweets 中的标点；另一方面，对于邮箱、网址，我们希望尽可能予以保留。

对于绝大部分特殊符号，直接将其剔除。而对于"?'./!"，考虑到它们可能包含的其他含义，例如可能在网址的 url 中出现。我们将每条'text' 按空格分词，将分词后每个词的开头和结尾的标点符号去掉，但是保留 @ 字符。

这一步返回按空格分词后的列表。

2.2.3 去除停用词

教材中给出了停用词列表，我们在这里直接使用。将停用词存储在一个.txt 文件中，读取该文件，并从上一步按空格分词后的列表中去除在此文件中的停用词。

2.2.4 词干还原

本实验采用的词干还原方法是 Porter 方法。通过 Python 的第三方库 nltk 来实现。Porter 方法有一个特点，会将同源的单词还原到相同的形式，但是不一定是正确的原形。

词干还原后，得到一个二维列表。该二维列表每一行的第一项为 ID，后面若干项为该 ID 对应文本包含的词项。最后将列表按照 ID 进行排序。

```
In [7]: inverted_index = get_inverted_index(data)
        inverted_index_file = open('inverted_index.txt','w',encoding='utf-8')
        inverted_index_file.write(str(inverted_index))
        inverted_index_file.close()
        print(inverted_index[12010:12020])
```

```
[['em', [24, 30140006722969600, 301710958420688896, 301835432755331072, 302137452007669761, 302455455777701890, 30280495124193280, 304397288481103872, 305444522333200385, 306374440806072 33, 307544274776510464, 309964690166001664, 311966509855547392, 313423107731890176, 3353921727 5486208, 33840094867623936, 34314860862898178, 34397493319962624, 62424989096558 1824, 62479032703191859 3, 62493595324164915 2, 62493724929416 8064, 62577681056392396 8, 62621730472923545 6, 62643073609937305 6]], ['email', [22, 29576869535817728, 29898299192385538, 30327181413672 9600, 30364671358182 6049, 30789408392020787 2, 30789459982380236 9, 30789862217390899 2, 30790264868059955 2, 30790426351280947 5, 30791248013336166 4, 30791433401993625 6, 30793603731269079 07, 30793752852106035 2, 30826155948916326 5, 30832130732866764 8, 30833357989990809 6, 30836798156035686 6, 30908041345876672, 31075112217021235 2, 31771605050855424, 62389077882623590 4, 62501442448356147 2]], ['eman', [2, 30067826060435456, 3006783 4360963073]], ['emanu', [3, 29615199631708161, 29748598148812800 0, 29846038121771100 80]], ['emanuel', [104, 29602019660533760, 29602145401569281, 29603137870364672, 29604188728074240, 29604332601090048, 29605381974007809, 29605434390220800, 29606415127547905, 29606637102702593, 2960853546907545 6, 29609861318705153, 29610484609060864, 29610644344934400, 29610756999749632, 29610900130373632, 29610928903299072, 296124192695255 04, 29616051847176192, 29616966620676096, 29617833742696448, 29620982062776320, 29621072722661376, 29621073565712384, 29622817754447872, 29628012471255040, 29630798625767425, 29631260305391616, 29634699332685824, 29635195187499009, 29637571675951104, 29642099909468160, 29642662654853 12, 29649212366917632, 29656497839415296, 29661336879239169, 29671069405151232, 29676981104680960, 29688185005019136, 29694510963679 36, 29708217516826624, 29714838418628608, 29716088212168705, 29720077087547392, 29726337899962368, 29732374158992998 4, 29736787173707776, 29753054228119553, 29816837160046592, 29873701256167424, 29891046645170177, 29894770717368320, 29914525234896897, 29938994254979072, 29942734030635008, 29944405947322368, 29950724490133505, 29963957057880067, 29966764259418112, 29979785870450690, 29979787090984960, 29981750247563264, 29986568634634241, 29992277275316225, 29995067955482624, 29996481968611328, 30002622148644866, 30008563728392192, 30015571043033088, 30015603104284672, 30022830624088064, 30097429839740928, 30101682566209536, 30233366276087808, 30305888401104896, 30307254150365184, 30356159202336768, 30429676585353216, 30764349832306689, 30767633527742465, 30768499169169410, 30769192563113984, 30769661930905601, 30769994920890369, 30770321975939072, 30773099582464000, 30777895618088960, 30778510737940482, 30784884444237826, 30786150633308160, 30787730694733825, 30793840872923137, 30794785610530816, 30796220553236482, 30806290825478144, 30826476634902528, 30828837197578241, 30841242736660480, 30846813036617728, 30850469203025920, 30882540759810048, 30933644491100161, 30993728348880896, 31018836186628097, 31185639047172097]], ["emanuel'", [11, 29964806458970112, 29968696868864000, 29976628511645697, 29981750247563264, 29988033105240064, 30013954906398720, 30018721376378881, 30252359724572672, 30272609513897984, 30792615909662720, 31072366792544257]], ["emanuel\\'", [1, 29967133987311617]], ['emanuel' ', [3, 29995290803044352, 30764163454214144, 30933644491100161]], ['emarket', [6, 30642168938889216, 30642236995674112, 30642254842433536, 30642412799922
```

图 3: 倒排索引切片

3 建立倒排索引

建立倒排索引大致可以归纳为两步：

第一步：对于每条 tweet，即词干还原后返回的列表的每一行，建立词项字典。使用 Python 的字典结构，key 为词项，value 为该条 tweet 的 ID。前面列表中出现的空词项，在这一步需要去除。

第二步：对每一行建立的词项字典进行合并。合并后的字典用嵌套列表来记录。该嵌套列表可以看作一个二维列表，每一行表示一个词项的相关信息，第一项为该词项，第二项为一个列表，记录出现该词项的频率及所有文档的 ID。由于词干还原后的列表是按 ID 排好序的，因此这里的嵌套列表词项 ID 是有序的，但是我们需要对词项进行排序，使所有词项按照字母序排列。

将建立好的倒排索引储存到文件中。

4 布尔查询

基于上面建立的倒排索引，本实验设计了一个简略的布尔查询框架，包括简单查询和高级查询。其中，高级查询支持特定情况下的查询优化。

4.1 前期准备

前期准备包括用户输入查询语句、读入倒排索引、数据预处理等。

用户可以直接选择查询功能，并输入查询语句。用户需按照正确的格式进行输入。系统会对用户输入的语句进行预处理。

为了防止每次查询都要重新生成倒排索引，一个策略是使用 Jupyter notebook，将代码分块运行，中间数据保存在内存中，无需每次重新生成。或者分成多个.py 文件运行，在查询时需要先对前面文件生成的倒排索引进行读取。同时为了求补，还需要对所有 tweets 进行读取。

对 quary 数据预处理的方式与 tweets 大致相同。为了避免不必要的麻烦 (停用词列表中有'and')，这里省略了去除停用词的一项。去除停用词的策略在这种较短的查询中没有太大必要。

数据预处理最终返回一个列表。对于一个单词的查询，列表中只有这一个元素；对于两个单词的查询，列表中包含三个元素：单词和逻辑连接词。这里将'not' 与其后面的单词视为一体，以便于后面的处理。

4.2 简单查询

简单查询指的至多含有一个逻辑连接词的查询，可以直接对单词进行查询，或者查询两个单词通过'and' 或者'or' 连接词连接的语句。若连接的单词前出现'not' 则考虑对该单词求补，以查询不包含该单词的 tweet。由于简单查询包含较多的模式，因此需要用条件语句进行判断。

数据预处理返回的列表长度可能为 1 或 3。列表长度为 1 时，只需要用该单词在倒排索引词项中查询，返回的 ID，再返回 ID 对应的 tweets。列表长度为 3 时，首先读取逻辑连接符，再根据逻辑连接符作进一步判断。

fig. 4 展示的是采用教材中的简单算法对表进行合并，包括求并和求交两种。

4.3 高级查询

高级查询包括多个简单查询，通过逻辑连接符将多个简单查询的结果做进一步的布尔运算。默认的运算顺序为从左到右，即先输入的先参与运算。而对于全'and' 的情况则是个特例。程序支持全'and' 的查询优化。由于布尔运算中可能还含有嵌套的'and'，我们需要将这些'and' 提取出来，检索用'and' 连接的所有词项，并按照出现频率从低到高取交集进行合并操作。

In [12]:
```
def list_op(list1,list2,op): #求两个有序列表的合并算法 交集或补集
    answer = []
    if op == 'and':
        i = j = 0
        while i<len(list1) and j<len(list2):
            if list1[i]==list2[j]:
                answer.append(list1[i])
                i+=1
                j+=1
            elif list1[i]<list2[j]:
                i+=1
            elif list1[i]>list2[j]:
                j+=1
    elif op == 'or':
        answer = list1[:] #要新建不要索引
        for j in list2:
            if j not in answer:
                answer.append(j)
        answer.sort()
    return answer
```

In [13]:
```
a = [2,3,6,9,10,11]
b = [1,2,6,7,9,10,11,13]
print(list_op(a,b,'and'))
print(list_op(a,b,'or'))
```

```
[2, 6, 9, 10, 11]
[1, 2, 3, 6, 7, 9, 10, 11, 13]
```

图 4: 表合并实现

In [*]:
```
#简单查询
query = input(' \n请输入您的查询语句: ')
query_list = query_preprocess(query)
qualified_quary_list = []
if len(query_list) == 1:   #简单查询
    qualified_id_list = basic_query(query_list[0],inverted_index)
    print_tweets(qualified_id_list, id_data)
elif len(query_list) == 3:    #简单布尔查询
    qualified_id_list = simple_bool_query(query_list[1],query_list[2],query_list[0],inverted_index,id_list)
    print_tweets(qualified_id_list, id_data)
else:
    print('输入异常!')
```

```
 请输入您的查询语句: Ron and Birthday
共找到符合条件的结果43条
['Happy birthday to my boy Ron Weasley']
['@YUNGRONBOI happy Birthday Ron!!!']
['Just found out that the fictional character Ron weasley is in fact my soul mate since his fake birthday makes him a fake Pisces!
s!!!!!']
['Happy Birthday Ron Weasley ;;)']
['HAPPY BIRTHDAY RON WEASLEY !!!']
["It's Ron Weasley's birthday! The ginger who vomited slugs out from his mouth' happy birthday Ron! #RonWeasleyBirthday"]
[' ;-* "@Potteristic: Happy Birthday to our King Ron Weasley! http://t.co/1frCK1cwGs" ']
['"@Lord_Voldemort7: Happy Birthday Ron Weasley! Proof that not even magic can\'t fix being a ginger. #WeasleyIsOurKing" #ProudOfT
['Set de fotos: March 1st 1980 - Happy 33rd Birthday Ron! http://t.co/cPWRQ7gnch']
["Why isn't happy birthday Ron weasley trending? :@"]
['AND... HAPPY BIRTHDAY TO RON WEASLEY!!!']
['#Weasleyisourking Happy Birthday Ron Weasley!']
['Happy Birthday to one of my favourite gingers out there- Ron Weasley! &lt;3']
["Almost forgot that today is Ron Weasley's birthday. Well, Happy B'day Ron. #ProudtobePotterhead #HappyBdayRonWeasley"]
["Me and @MichelinaScotti just reunited on Ron Weasley's birthday. Coincidence? No."]
```

图 5: 简单查询演示

In [20]:
```
#高级查询
bool_query_num = int(input('请输入高级查询包含的简单查询数量：'))
simple_query_list = []
logical_operator_list = []
qualified_complex_id_list = []
for i in range(bool_query_num):
    simple_query_list.append(query_preprocess(input('请输入第{}个简单查询：'.format(i+1))))
    if i < bool_query_num-1:
        x = input('请选择第{}和{}个简单查询之间的逻辑连接词（1:and 2:or)'.format(i+1,i+2))
        if x=='1':
            logical_operator_list.append('and')
        elif x=='2':
            logical_operator_list.append('or')
        else:
            print('输入错误，默认为"and"')
            logical_operator_list.append('and')
print(simple_query_list)
print(logical_operator_list)
qualified_complex_id_list = complex_bool_query(simple_query_list,logical_operator_list,inverted_index,id_list)
print_tweets(qualified_complex_id_list, id_data)
```

```
请输入高级查询包含的简单查询数量：3
请输入第1个简单查询：Ron and birthday
请选择第1和2个简单查询之间的逻辑连接词（1:and 2:or)1
请输入第2个简单查询：Boys or girls
请选择第2和3个简单查询之间的逻辑连接词（1:and 2:or)1
请输入第3个简单查询：Happy
[['and', 'ron', 'birthday'], ['or', 'boy', 'girl'], ['happi']]
['and', 'and']
共找到符合条件的结果1条
['Happy birthday to my boy Ron Weasley']
```

图 6: 复杂查询演示

5 心得与总结

通过本实验，我复习了很多的 Python 基础语法和编程技巧。包括对文件的读取、列表字典等结构的相关方法。同时也加深了对倒排索引和布尔查询的理解。

由于自己学习 Python 的时间也并不长，对一些原理的理解也不够深刻。附件 bool_query.ipynb 为实验代码。代码一定不是完美的，不论是在可读性、时间复杂度等方面，都有较大的进步空间。但是根据教材内容结合自身理解来实现原理，就是最好的学习方式。