

Lab1-实验报告

钟溯颍 PB18111764

单航 PB18111747

代码描述

tokenization_opt.py

这个代码在utils文件夹里，读取path文件，用来生成倒排表（前一千）

1. 首先利用正则化过滤文件（邮件头部，邮箱地址，URL，HTML标签），减小分词工作量以及优化索引效果
2. 接着迭代读取每个文档(try打不开的就不管他了，视为空)，进行如下操作：
3. 使用nltk包进行分词，去停用词，使用SnowballStemmer进行词根化，存到同名文件中
 - 在一开始并没有选择将其存储，但考虑到建立矩阵时还需要用到，为了保证一致性便选择存储。
4. 直接进行倒排索引的构建，把索引看成是一个字典<token, [docID, docID2, ...]>，对每个词进行遍历：
 - 若某词项有对应key，则看对应列表的最后一个有无该词[优化](#)，没有则append进去
 - 若某词项没有，则给该词项建立初始为此时的docID的列表项
5. 输出倒排表为index.txt，在迭代时，将其输出为间距格式[优化](#)，同时还输出df.txt即词项的df值。

bool_search.py

这个代码在src文件夹中，获取Bool查询的输入，用来输出符合规则的相应文档：

1. 首先是arg_parse获取输入，以下是一开始的蠢想法：

考虑到群里的要求，需要做一个语义分析。

根据，OR，AND，NOT由低到高的优先级，一个bool检索可写成如下文法。

$E \rightarrow E \text{ OR } T \mid T; T \rightarrow T \text{ AND } F \mid F; F \rightarrow \text{NOT } F \mid (E) \mid \text{token}$

最后发现，将布尔表达式的中缀式换成后缀式即可

以上的实现写在 `parse_query()` 中


2. 其中，函数意义分别如下：

- `parse_query()` 用来将表达式正则化，词根化，并转化为后缀式，以列表的形式存储。
- `load_index()` 从倒排文件中，找到相对应的词项及其文档表，以列表的形式返回
 - 直接遍历获取，没什么说的
 - 把间距变回正常序号
- `bool_AND()`, `bool_OR()`, `bool_NOT()` 分别是三个不同操作符的实现
 - AND是一个个比对，保留相同的。使用书上的合并算法（没有跳表指针，因为没要求

- OR是一个个比对，有就留下来，注意一个表遍历完了还要继续遍历没完的。
- NOT是直接对这个表求反，即返回所有DOC集合没有这个文档的
- `search()` 是对 `parse_query()` 得到的操作符和词项，存入栈中，进行操作，并返回bool查询的最终结果
 - 即是后缀表达式的计算规则
 - 迭代输入，如果是词项，放进栈中；如果是AND或OR，则对栈头两个元素操作，结果放回栈里；如果是NOT，则只对一个元素进行操作。知道遍历完输入，吐出栈底元素。

3. 通过 `search()` 获得的即为文档列表，并对其包装，输出


- 第一行是原表达式的输出
- 第二行是表达式在栈中的后缀表达式形式
- 第三行往后是提示在第几个倒排项找到了某个词
- 以及布尔查询结果的输出(通常很长)，如图及索引结果验证

exp1 > output >  bool_debug.log

- ```

1 power&businessORenergy.& natural.AND.signed
2 ['power', 'busi', '&', 'energi', 'natur', 'sign', '&', '&', '|']
3 power
4 successfully find power in invert index No 3 !
5 busi
6 successfully find busi in invert index No 34 !
7 energi
8 successfully find energi in invert index No 4 !
9 natur
10 successfully find natur in invert index No 188 !
11 sign
12 successfully find sign in invert index No 239 !
13
14
15 result is docID = 16 which is maildir/arnold-j/notes_inbox/67.
16
17 result is docID = 23 which is maildir/arnold-j/notes_inbox/45.
18
19 result is docID = 50 which is maildir/arnold-j/notes_inbox/56.
20


```

exp1 > dataset > maildir > arnold-j > notes\_inbox >  67.

- ```

184 art giving more > power 9 of 69
185 vances, Houston may owe a great deal to the study of small substan
186 notechnology is the study of creating functional structures on a m
187 ale (the prefix "nano" means one billionth, or 10 to the ninth pow
188 merically). Its theories and practices give scientists the means t
189 nstruct useful entities using the smallest known particle of unalt

```

exp1 > dataset > maildir > arnold-j > notes_inbox >  67.

- ```

729 ar as po > signed 1 of 3
730 s to tak
731 ple at one time there were more than 200 MoUs signed up
732 ects, the government provided counter guarantees for

```

## matrix.py

代码在utils文件夹里，用来生成tf-idf矩阵

1. 首先获取文档数目，然后打开df.txt读取词项-DF对，并计算idf
2. 遍历之前存储的分词后的文档，存储tf值。
3. 根据tf-idf公式  $W_{t,d} = (1 + \log tf_{t,d}) \cdot \log N/df_t$ ，循环填入矩阵
4. 使用 `scipy.sparse.csr_matrix` 按行压缩并使用 `save_npz` 存储

## semantic\_search.py

代码在src文件夹里，用来计算相似度并排序

1. 老样子使用 `argparse` 来获取参数输入，并用 `sparse.load_npz` 读取压缩文件（很耗时）
  - 打印了 `head()`，可以看到tf-idf的大概样子
2. 使用 `split()` 和词根化获取输入，并根据token位置存成向量的形式
3. 使用余弦相似度对查询向量与tf-idf向量迭代进行运算，并将结果存放在 `result` 里面
4. 使用 `pandas` 进行排序，并包装结果进行输出：

使用查询“The president plans to be a power and energy company” 结果如图：

```
Length: 1000, dtype: float64

[63257 62388 65022 97579 97797 97329]
The Top 10 answers and scores are:
maildir/forney-j/sent_items/154.
with score: 0.4472135954999579
maildir/mcconnell-m/sent/372.
with score: 0.5568624242124827
maildir/mcconnell-m/_sent_mail/371.
with score: 0.5568624242124827
maildir/mcconnell-m/all_documents/587.
with score: 0.5568624242124827
maildir/tholt-j/sent_items/6.
with score: 0.5341333502777612
maildir/tholt-j/sent/5.
with score: 0.5341333502777612
maildir/tholt-j/_sent_mail/5.
with score: 0.5341333502777612
maildir/tholt-j/all_documents/118.
with score: 0.5341333502777612
```

372.

Message-ID: <18348372.1075843965785.JavaMail.evans@thyme>  
Date: Wed, 29 Nov 2000 17:54:00 -0800 (PST)  
From: mike.mcconnell@enron.com  
To: david.terlip@enron.com  
Subject: Re: The New Power Company  
Mime-Version: 1.0  
Content-Type: text/plain; charset=us-ascii  
Content-Transfer-Encoding: 7bit  
X-From: Mike McConnell  
X-To: David A Terlip  
X-cc:  
X-bcc:  
X-Folder: \Mark\_McConnell\_June2001\Notes Folders\Sent  
X-Origin: MCCONNELL-M  
X-FileName: mmcconn.nsf  
  
Thanks.  
m

```
INAL PROBLEMS OUTPUT DEBUG CONSOLE
e) → src git:(main) ✕ open ../dataset/maildir/mcconnell-m/sent/372.
e) → src git:(main) ✕
```

# 结果优化

## 1.索引的时空复杂度的优化

时间优化： 之前用以下python式语句判断是否添加倒排表词项：

```
if docID not in invertIndex[token]:
```

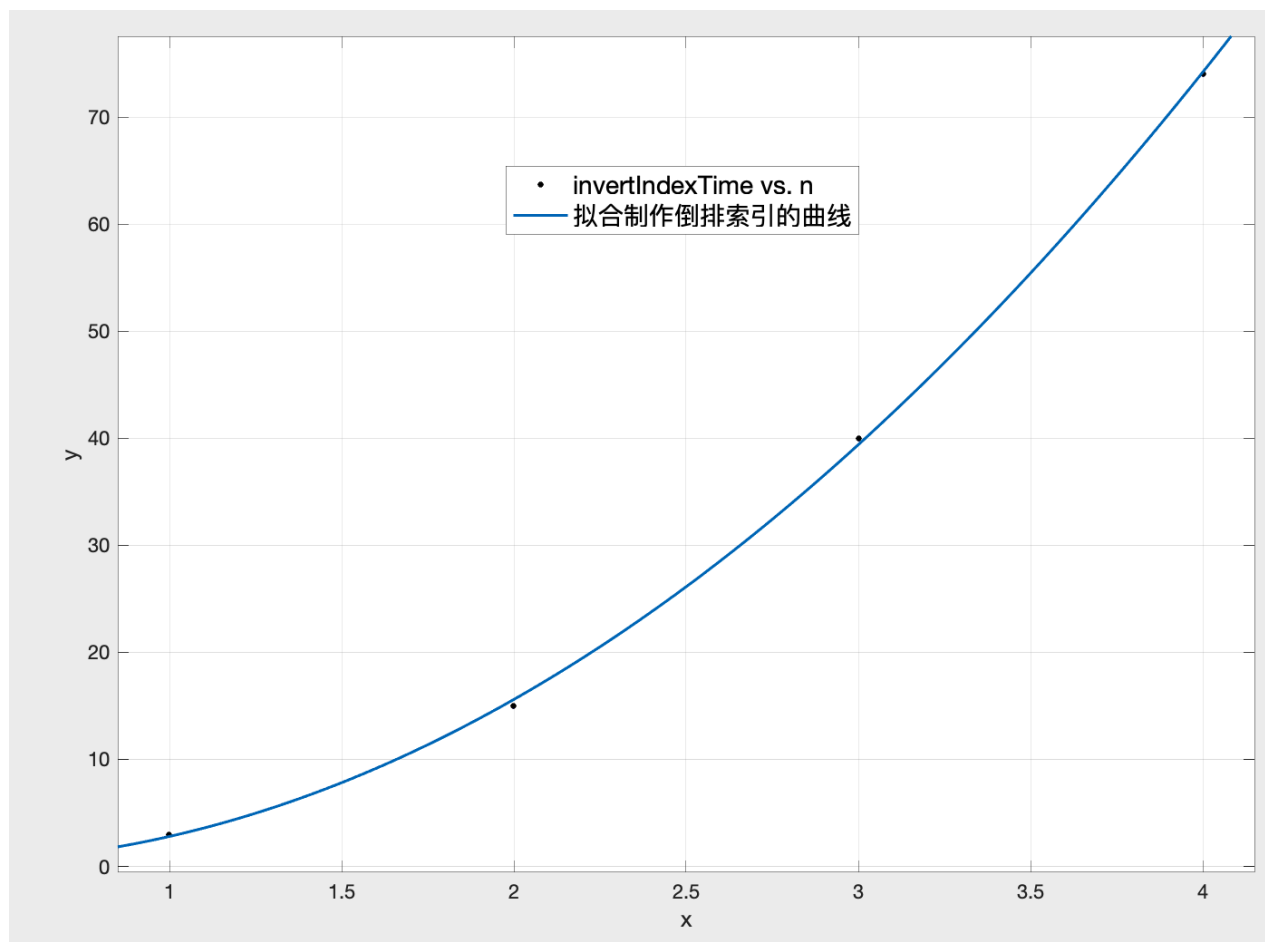
这导致时间复杂度与文档数量不成线性关系，是 $O(n^2)$ 的，输出小规模耗时如图及拟合图：

```
total·docs·:=·10000
total·time·:=·27.230363845825195
index·time·:=·3.009058952331543
reTime·:=·3.4250636100769043
stopTime·:=·0
stemTime·:=·20.05732035636902
```

```
total·docs·:=·30000
total·time·:=·146.12736988067627
index·time·:=·40.184300899505615
reTime·:=·14.366731882095337
stopTime·:=·0
stemTime·:=·78.54025053977966
```

```
total·docs·:=·20000
total·time·:=·80.02519822120667
index·time·:=·15.990978956222534
reTime·:=·8.767294645309448
stopTime·:=·0
stemTime·:=·47.34056520462036
```

```
total·docs·:=·40000
total·time·:=·216.53626799583435
index·time·:=·74.12460947036743
reTime·:=·19.39746117591858
stopTime·:=·0
stemTime·:=·103.91469812393188
```





这导致若是大规模进行生成，需要数小时，很不现实。于是做了一个修改：

```
if invertIndex[token][-1] != docID :
```

这就使得时间大幅降低：

|    |                                                                                                                                                                    |     |                                                                                                                                                                    |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 原： | <pre>total·docs·=·40000 total·time·=·216.53626799583435 index·time·=·74.12460947036743 reTime·=·19.39746117591858 stopTime·=·0 stemTime·=·103.91469812393188</pre> | 改进： | <pre>total·docs·=·40000 total·time·=·131.87948274612427 index·time·=·4.242624044418335 reTime·=·18.004252910614014 stopTime·=·0 stemTime·=·97.88430166244507</pre> |
|----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|

最终耗时(s)为 `total·time·=·2104.855010986328` ，（若存储分词结果则为 `2743.555778026581`）

空间优化：

与讲义不同，并不是读取所有文档后再进行建立索引，那样至少会占用所有文档大小的内存。本实验采用迭代文档分别进行分词，词根化，添加进倒排表的操作。并对其进行了如上的时间优化。

存储也做了优化：在一开始，输出，即保存倒排表，这里用了很蠢的方法，即直接保留ID：

invert\_20w.txt

今天 下午10:47

74.1 MB

纯文本文稿

invert\_20w.txt

power [5, 16, 23, 28, 46, 50, 55, 78, 102, 117, 142, 147, 153, 155, 166, 168, 169, 171, 172, 178, 195, 199, 212, 223, 232, 241, 245, 246, 251, 252, 261, 263, 273, 286, 287, 297, 305, 308, 312, 317, 323, 326, 335, 343, 353, 358, 366, 392, 398, 419, 429, 447, 456, 463, 466, 469, 479, 481, 490, 499, 503, 505, 509, 517, 518, 524, 533, 541, 542, 543, 545, 552, 556, 572, 577, 582, 589, 591, 594, 612, 616, 623, 624, 630, 633, 641, 653, 663, 669, 670, 676, 679, 683, 684, 702, 703, 714, 755, 759, 790, 791, 809, 846, 854, 855, 877, 881, 898, 900, 903, 908, 915, 916, 946, 949, 965, 978, 984, 986, 992, 1012, 1042, 1080, 1088, 1095, 1097, 1099, 1101, 1123, 1139, 1152, 1153, 1204, 1240, 1261, 1292, 1318, 1326, 1348, 1358, 1444, 1465, 1471, 1477, 1498, 1523, 1524, 1532, 1550, 1568, 1575, 1576, 1577, 1620, 1629, 1632, 1636, 1648, 1661, 1685, 1722, 1727, 1766, 1795, 1797, 1859, 1896, 1901, 1911, 1926, 1929, 1957, 1966, 1991, 2029, 2035, 2038, 2128, 2161, 2173, 2174, 2236, 2304, 2363, 2372, 2395, 2404, 2409, 2424, 2475, 2497, 2498, 2504, 2511, 2522, 2541, 2545, 2561, 2565, 2596, 2641, 2644, 2662, 2669, 2697, 2719, 2741, 2752, 2777, 2783, 2814, 2829, 2841, 2859, 2873, 2890, 2900, 2930, 2937, 2942, 2952, 2963, 2974, 2995, 2996, 3003, 3006, 3055, 3062, 3072, 3090, 3095, 3114, 3120, 3165, 3222, 3256, 3275, 3281, 3298, 3327, 3384, 3434, 3443, 3444, 3451, 3456, 3464, 3465, 3469, 3479, 3487, 3498, 3503, 3504, 3507, 3522, 3594, 3635, 3652, 3663, 3686, 3720, 3728, 3748, 3790, 3833, 3880, 3881, 3915, 3953, 3963, 3971, 3990, 3991, 3992, 4027, 4040, 4050, 4059, 4064, 4077, 4103, 4123, 4150, 4151, 4154, 4173, 4218, 4223, 4270, 4283, 4323, 4326, 4347, 4354, 4367, 4375, 4382, 4394, 4397, 4426, 4491, 4511, 4523, 4582, 4603, 4635, 4639, 4646, 4647, 4649, 4653, 4664, 4675, 4688, 4726, 4732, 4735, 4744, 4782, 4800, 4824, 4858, 4888, 4900, 4901, 4902, 4904, 4905, 4906, 4907, 4914, 4918, 4924, 4926, 4927, 4931, 4933, 4942, 4949, 4958, 4980, 4986, 4988, 4991, 5000, 5015, 5020, 5029, 5051, 5058, 5074, 5077, 5087, 5089, 5091, 5094, 5104, 5108, 5118, 5119, 5127, 5128, 5131, 5136, 5143, 5144, 5147, 5166, 5185, 5193, 5196, 5198, 5201, 5205, 5207, 5221, 5223, 5224, 5225, 5232, 5241, 5243, 5246, 5259, 5263, 5273, 5275, 5285, 5290, 5310, 5313, 5315, 5322, 5329, 5331, 5334, 5342, 5355, 5357, 5362, 5364, 5371, 5373, 5383, 5386, 5394, 5396, 5410, 5417, 5418, 5427, 5428, 5437, 5438, 5439, 5440, 5446, 5455, 5456, 5462, 5466, 5467, 5468, 5469, 5476, 5477, 5480, 5481, 5487, 5491, 5493, 5497, 5500, 5501, 5502, 5508, 5509, 5517, 5531, 5535, 5536, 5543, 5550, 5551, 5556, 5558, 5560, 5564, 5565, 5568, 5580, 5583, 5584, 5586, 5588, 5589, 5591, 5592, 5606, 5615, 5618, 5621, 5629, 5632, 5633, 5641, 5643, 5656,

可以看到才20w就已经74MB了，所以才用了存储间距的方法：最后50w文档生成的大小为：

index.txt

昨天 下午1:56

60.5 MB

纯文本文稿

以上。