

0. 提示

本文前两个部分比较无聊，只想看图的同学可以直接跳到全文末尾

1. 问题来源

暑假的时候，闲的没事学了一下python，为了练手，决定做一个北大树洞的可视化分析。进行完了比较简单的一些消息频率、树洞热度之类的数据分析以后，觉得仅仅呈现了一下数据有些无聊，所以希望能做一些更有趣的东西出来。

首先想到的一个就是关键词分析，也就是给出一个查询词，能够知道人们谈论它的时候，还会提到些什么其他的关键词。如果我搜索某门通选课的时候，能返回一大堆人们对它的评价，这岂不是很好？另外也可以方便的了解同学们对一些事情的态度和看法，例如我搜索“保卫部”的时候，就会返回一大堆大家对保卫部的看法。虽然说也没什么实际用处，但总归还是挺有趣。

由于水平所限，我也只能从一些简单的思路出发。首先给出一个查询词，并搜索出所有包含有这一查询词的树洞消息集合，下一步要做的，则是从这一集合中提取出最有用的一些关键词来。

当时我采用的是TF-IDF算法，这一算法的主要思想是：字词的重要性随着它在文件中出现的次数成正比增加，但同时随着它在语料库中出现的频率成反比下降。给出一系列可能成为关键词的词语，依次计算它们的TF-IDF值，选出最大的几个就好了。当时基于这一算法也的确做出了一些有趣的东西。

当时我曾注意到了一个叫什么什么rank的算法，跟TF-IDF在一起，都出现在了jieba分词库里。简单了解了一下发现这个算法是基于图论的，然而那个时候我并没有学过图论，所以也没太在意这是个什么东西。

这个学期在《人群与网络》课上学过了PageRank算法之后，我发现这一算法的用处其实远不止互联网中的链接分析。《网络、群体与市场》课本第14.5节中给出了一个用PageRank来计算科技期刊影响因子的例子，给出了一个美国最高法院引用连接分析的例子；此外，在网上简单的查一下PageRank，还会发现这个东西还能用来做微博用户分析、wikipedia编辑者分析等等。

当你知道PageRank可以用来分析用户之后，你甚至会想到能不能用PageRank来做那个微信好友推测的期末大作业，比如算一算全部节点的PageRank值，作为一个衡量指标考虑进去。当然这个我还没试过，有兴趣的同学可以试一试告诉我效果，如果好的话我也做一下。

非常自然地，我又一次想到了暑假时的那个问题——PageRank能不能用来做关键词抽取呢？事实上这个应该是实现的，只要合适地定义一下什么叫做“词与词之间的链接”就好了，至于效果好不好反正我也没试过也不知道。这时我依稀的想起了当初那个什么什么rank算法，有种不祥的预感，于是赶紧去查了一下，发现那个“什么什么rank”，正是TextRank算法。

2. 由PageRank到TextRank

2.1 PageRank算法简介

请翻开《网络、群体与市场》p.253页自行复习课本14.3节，或参阅[Wikipedia相关内容](#)。

2.2 PageRank算法的概率意义

设 $G = \langle V, E \rangle$ 是一个有向图， V 为节点集合， E 为边集合。对于每一个节点 V_i ， $In(V_i)$ 表示指向它的节点， $Out(V_i)$ 表示由它指向的节点，则 V_i 的PageRank分数为：

$$S(V_i) = (1 - d) + d * \sum_{j \in In(V_i)} \frac{1}{|Out(V_j)|} S(V_j)$$

其中 d 是损耗因子，通常设置为0.85。

由PageRank的概率意义，基于随机游走模型，可以提出以下等价计算式。设图中节点数为 n ，设列向量 V_0 是初始状态下各节点的概率分布(均为 $1/n$)， V_i 是第 i 次迭代后的各节点概率分布。同时构造一个转移概率矩阵 $M_{n \times n}$ ， M_{ij} 表示由第 j 个节点转移到第 i 个节点的概率，则有：

$$V_{i+1} = dMV_i + (1 - d) * V_0$$

上式将PageRank的迭代计算表示成了矩阵运算，在面对大规模数据时，采用稀疏矩阵等优化方法可以极大提高运算效率。循环迭代直到 V_i 收敛，即可得到各个节点的PageRank值。

2.3 词共现

从概率的意义出发，我们下面需要考虑的是，如何表示两个词之间的转移概率。其中一种简单的方法是计算词共现(co-occurrence)指数。如果两个词语在某个长度为 N 的窗口中同时出现，我们就称这两个词有链接， N 通常取2-10个词。两个词语之间的词共现指数，可以定义为它们间的链接数目。

我们之前已经筛选出了所有包含查询词的树洞消息集合，下面只需要对集合中的各个消息进行分词处理，并计算各个词之间的词共现指数即可。

因为词共现并没有表现出两个词之间“谁指向谁”的关系，因此只能构造出一个无向图来，将无向图的每条边都替换成两条反向的边就可以得到一个有向图。类比于链接分析，词a和词b的词共现指数，就可以理解是a指向b的链接数，同时也是b指向a的链接数。

2.4 算法实现

上文中已经提及，实际上TextRank算法在jieba分词包里面已经有了。此处也不打算介绍jieba是如何处理文本数据的，有兴趣的同学可以去自行查阅[jieba源码](#)。

仔细阅读发现，jieba的TextRank算法实现的其实比较粗糙。jieba自己造了个图类，只进行了10次迭代，计算甚至有可能还没有收敛就停止了。而且jieba的算法是用python的循环语句来运算的，没有使用numpy等高效的矩阵运算库，可能很难处理大规模数据。(所以jieba限制了10次循环，虽然数据量大，但是人家跑的少啊)

随后的实践证明，即使改用矩阵运算实现，如果不使用稀疏矩阵，当文本长度达到数千字之后，程序也已经很难跑下去了，恐怕处理数百MB的语料就会对内存造成很大的压力。

造了半天轮子之后，由于水平、时间、精力有限，我也没再研究如何用稀疏矩阵或者用其他的方法进一步优化了，而是发现了networkx库中有[现成的pagerank算法](#)。networkx库中的算法实现的比较完善，不仅使用了高效的矩阵运算方法，还支持设置更多的矩阵特征，更重要的是，使用方法极其简单，一行代码就解决了所有问题：

```

1 import networkx as nx
2 nodes_rank = nx.pagerank_numpy(graph)
3 # 或者 nodes_rank = nx.pagerank_scipy(graph)
4 # 或者 nodes_rank = nx.pagerank(graph)

```

使用jieba的方便之处是省去了自己处理文本数据的麻烦，直接交给jieba就好了，不过需要注意手动设置停止词。下面附上使用networkx优化之后的jieba中的TextRank算法，有兴趣的同学可以直接用它把jieba库里原有的textrank.py文件替换掉，再装好networkx包，就可以使用进阶版TextRank算法了，使用方法完全不变。

```

1 #!/usr/bin/env python
2 # -*- coding: utf-8 -*-
3
4 from __future__ import absolute_import, unicode_literals
5 import sys
6 from operator import itemgetter
7 from collections import defaultdict
8 import jieba.posseg
9 from .tfidf import KeywordExtractor
10 from ..compat import *
11 import networkx as nx
12
13 class TextRank(KeywordExtractor):
14
15     def __init__(self):
16         self.tokenizer = self.postokenizer = jieba.posseg.dt
17         self.stop_words = self.STOP_WORDS.copy()
18         self.pos_filt = frozenset(('ns', 'n', 'vn', 'v'))
19         self.span = 5
20         self.g = None
21         self.edges = []
22
23
24     def pairfilter(self, wp):
25         return (wp.flag in self.pos_filt and len(wp.word.strip()) >= 2
26                 and wp.word.lower() not in self.stop_words)
27
28     def textrank(self, sentence, topK=20, withWeight=False, allowPOS=
29 ('ns', 'n', 'vn', 'v'), withFlag=False):
30         """
31         Extract keywords from sentence using TextRank algorithm.
32         Parameter:
33             - topK: return how many top keywords. `None` for all
34             possible words.
35             - withWeight: if True, return a list of (word, weight);
36                           if False, return a list of words.
37             - allowPOS: the allowed POS list eg. ['ns', 'n', 'vn',
38             'v'].

```

```

36         if the POS of w is not in this list, it will be
filtered.
37         - withFlag: if True, return a list of pair(word, weight)
like posseg.cut
38         if False, return a list of words
39         """
40         self.pos_filt = frozenset(allowPOS)
41         self.g = nx.Graph()
42         cm = defaultdict(int)
43         words = tuple(self.tokenizer.cut(sentence))
44         for i, wp in enumerate(words):
45             if self.pairfilter(wp):
46                 for j in xrange(i + 1, i + self.span):
47                     if j >= len(words):
48                         break
49                     if not self.pairfilter(words[j]):
50                         continue
51                     if allowPOS and withFlag:
52                         cm[(wp, words[j])] += 1
53                     else:
54                         cm[(wp.word, words[j].word)] += 1
55
56
57         for terms, w in cm.items():
58             tmp = (terms[0], terms[1], w)
59             self.edges.append(tmp)
60         self.g.add_weighted_edges_from(self.edges)
61
62
63         nodes_rank = nx.pagerank_numpy(self.g)
64         if withWeight:
65             tags = sorted(nodes_rank.items(), key=itemgetter(1),
reverse=True)
66         else:
67             tags = sorted(nodes_rank, key=nodes_rank.__getitem__,
reverse=True)
68
69         if topK:
70             return tags[:topK]
71         else:
72             return tags
73
74         extract_tags = textrank
75

```

3. 结果示例

上面啰嗦了一大堆，终于到了看例子的时候了。

从martinwu42做的[树洞备份项目](#)中，我们可以比较容易地获得近一段时间树洞的实时数据，或者以往的历史记录。样例如下：

```
1 #p 533330 2018-11-07 18:32:04 1 0
2 想给女朋友买件大衣（风衣？） 什么牌子好呀 有洞友有经验么QAQ
3
4 #p 533329 2018-11-07 18:30:23 1 0
5 地学楼的小教室门禁修好了吗？可以正常用了吗
6
7 #p 533328 2018-11-07 18:28:45 1 0
8 wzn经原出成绩了.....
9 心态已崩qwq 大家都怎么样qwq
10
11 #p 533327 2018-11-07 18:27:56 1 0
12 求问有没有同学认识尹泽尔这个同学呀？有一个包裹写错电话号码寄到我这里了
13
14 #p 533326 2018-11-07 18:25:26 1 0
15 北大中心馆在哪里呀？
16
17 #p 533325 2018-11-07 18:21:38 1 0
18 有偿求ymj经原今天作业的答案
```

此处获取了11月5日到11月7日晚上的数据进行分析，部分代码如下：

```
1 from jieba.analyse.textrank import TextRank
2
3
4 def load_data(file_path):
5     """
6     :param file_path: 树洞数据文件路径
7     :return: 返回一行一行的树洞数据，格式为list
8     """
9     with open(file_path, 'r') as f:
10         lines = f.readlines()
11         texts = []
12         for line in lines:
13             if "#p" not in line and "#c" not in line:
14                 texts.append(line.replace('\n', ' '))
15         return texts
16
17
18 def get_text_rank(texts,
19                    stopwords_path="/Users/yanjin/PycharmProjects/NLP/corpus/stopwords.txt"
20                    ):
21     """
22     :param texts: 之前拆分好的，一行一行的树洞数据，格式为list
23     :param stopwords_path: 分词时使用的停止词的目录
24     :return: 一个list，元素为tuple，存有词语和其textrank值
```

```

23     """
24     text = ""
25     for t in texts:
26         text += t
27     trk = TextRank()
28     trk.set_stop_words(stopwords_path)
29     ranks = trk.textrank(text, topK=100, withWeight=True)
30     return ranks
31
32
33 if __name__ == '__main__':
34     data = load_data('/Users/yanjin/Desktop/pkuhole.txt')
35     ranks = get_text_rank(data)
36     for i in ranks:
37         print(i[0], ': ', i[1])
38

```

由此得到的前100条数据如下所示，其中已经人为地删去了部分停止词。

```

1 同学 : 0.007496206910452123
2 老师 : 0.00679745606555012
3 喜欢 : 0.0049711891672249
4 时间 : 0.004594613177520519
5 女生 : 0.004201625700722834
6 作业 : 0.003789952834596567
7 学校 : 0.003690458347856126
8 工作 : 0.0034406962038228206
9 希望 : 0.003188378564289968
10 推荐 : 0.0030765027264502026
11 退课 : 0.002876528868949646
12 不想 : 0.002790585268909524
13 学习 : 0.0026283445331818775
14 考试 : 0.0024265897708600974
15 影响 : 0.0023762415654002235
16 发现 : 0.002271368356679058
17 学生 : 0.00227095921173799
18 中国 : 0.002228395065337381
19 大学 : 0.0021750604974747688
20 哥哥 : 0.002113486742181936
21 好像 : 0.002050472990660624
22 选择 : 0.0020491661755145728
23 男朋友 : 0.00202307443929152
24 找到 : 0.0019876708934543994
25 室友 : 0.0019024949054083744
26 想要 : 0.001896319118507025
27 男生 : 0.001893668769188421
28 微信 : 0.0018494885776057031
29 朋友 : 0.001842735698290266
30 相关 : 0.0018411179661677172

```

31	内容 :	0.0018122185246003712
32	公司 :	0.0017952631148812736
33	文化 :	0.0017724414531451172
34	原价 :	0.0017612862015933498
35	学长 :	0.0017590957305467884
36	学姐 :	0.0016965424846817506
37	宿舍 :	0.0016935905885195359
38	上课 :	0.0016796882025863218
39	调整 :	0.0016530679260576128
40	交换 :	0.001623667366829336
41	申请 :	0.0016232718042183442
42	学期 :	0.0016107525533183948
43	地方 :	0.001587081935539713
44	事情 :	0.0015486826314336168
45	有偿 :	0.0014861751374074395
46	妹子 :	0.0014797503918333467
47	生活 :	0.0014626020902104252
48	小时 :	0.001449712674141344
49	实习 :	0.0014427004304406932
50	经验 :	0.0014381510168737634
51	信科 :	0.0014323115312550806
52	吃饭 :	0.0014272642426834926
53	贵校 :	0.0014169867558218425
54	感谢 :	0.0014056750984009353
55	校园卡 :	0.001402250746818743
56	图书馆 :	0.0013740975989951524
57	信息 :	0.0013644614431161536
58	北京 :	0.001360177775850668
59	经历 :	0.0013477775357030788
60	成绩 :	0.0013465236264011994
61	小伙伴 :	0.0013276759791449222
62	教室 :	0.001321884109627044
63	中心 :	0.001306699181519923
64	好看 :	0.0012967798331068544
65	口罩 :	0.0012879686376073222
66	世界 :	0.0012810843938200881
67	自习 :	0.001274803796142643
68	组队 :	0.001238569742649441
69	照片 :	0.0012378931684407229
70	成绩单 :	0.0012366503083461303
71	专业 :	0.0012350425797586218
72	保研 :	0.0012142739294736266
73	院系 :	0.0012122132349647028
74	情况 :	0.0011981848666231172
75	留学生 :	0.0011883208498597478
76	心理 :	0.0011835857394157188
77	高考 :	0.001181592577519386
78	手机 :	0.0011731529345651848
79	购于 :	0.001164735971328065

