26 新词发现的信息熵方法与实现

Oct By 苏剑林 | 2015-10-26 | 36308位读者 引用

在本博客的前面文章中，已经简单提到过中文文本处理与挖掘的问题了，中文数据挖掘与英语同类问题中最大的差别是，中文没有空格，如果要较好地完成语言任务，首先得分词。目前流行的分词方法都是基于词库的，然而重要的问题就来了：词库哪里来？人工可以把一些常用的词语收集到词库中，然而这却应付不了层出不穷的新词，尤其是网络新词等——而这往往是语言任务的关键地方。因此，中文语言处理很核心的一个任务就是完善新词发现算法。

新词发现说的就是不加入任何先验素材，直接从大规模的语料库中，自动发现可能成词的语言片段。前两天我去小虾的公司膜拜，并且试着加入了他们的一个开发项目中，主要任务就是网络文章处理。因此，补习了一下新词发现的算法知识，参考了Matrix67.com的文章《互联网时代的社会语言学：基于SNS的文本数据挖掘》，尤其是里边的信息熵思想，并且根据他的思路，用Python写了个简单的脚本。

程序的算法完全来自于Matrix67.com的文章，感兴趣的读者可以移步到他的博客仔细阅读，相信会获益匪浅的。在此主要简单讨论一下代码的实现思路。具体程序请见文末。为了处理更大的文本，尽量不用Python内置的循环，尽可能用到第三方库所带的函数，如Numpy、Pandas。由于涉及到词语数量较多，要做好索引工作，比如词语先要排好序，会大大提高检索的速度。

下面是用本文程序对金庸小说《天龙八部》世纪新修版（txt电子版2.5M）做的新词发现结果（部分），用时20秒左右，感觉结果还不错，主要角色的人名都自动发现了。当然因为代码比较短，没做什么特殊处理，还有很多可以改善的地方。

```
段誉,3535
什么,2537
萧峰,1897
自己,1730
虚竹,1671
乔峰,1243
阿紫,1157
武功,1109
阿朱,1106
姑娘,1047
笑道,992
咱们,832
师父,805
如何,771
如此,682
大理,665
丐帮,645
突然,640
王语嫣,920
慕容复,900
段正淳,780
```

```
木婉清,751
鸠摩智,600
游坦之,515
丁春秋,463
有什么,460
包不同,447
少林寺,379
保定帝,344
马夫人,324
段延庆,302
乌老大,294
不由得,275
王夫人,265
为什么,258
只听得,255
是什么,237
云中鹤,236
那少女,234
巴天石,230
王姑娘,227
忽听得,221
钟万仇,218
少林派,216
叶二娘,216
朱丹臣,213
风波恶,209
契丹人,208
南海鳄神,485
慕容公子,230
耶律洪基,189
六脉神剑,168
站起身来,116
带头大哥,103
这几句话,100
点了点头,96
星宿老怪,92
神仙姊姊,90
吃了一惊,87
大吃一惊,86
慕容先生,86
又有什么,86
```

完整代码（**3.x**版本，简单修改可以用于**2.x**版本，主要是输出函数不同而已）：

```
import numpy as np
import pandas as pd
import re
from numpy import log,min

f = open('data.txt', 'r') #读取文章
s = f.read() #读取为一个字符串

#定义要去掉的标点字
drop_dict = [u', ', u'\n', u'。', u'、', u': ', u'(', u')', u'[', u']', u'.', u',', u'
for i in drop_dict: #去掉标点字
    s = s.replace(i, '')

#为了方便调用，自定义了一个正则表达式的词典
myre = {2:'(..)', 3:'(...)', 4:'(....)', 5:'(.....)', 6:'(......)', 7:'(.......)'}

min_count = 10 #录取词语最小出现次数
min_support = 30 #录取词语最低支持度，1代表着随机组合
min_s = 3 #录取词语最低信息熵，越大说明越有可能独立成词
max_sep = 4 #候选词语的最大字数
t=[] #保存结果用。

t.append(pd.Series(list(s)).value_counts()) #逐字统计
tsum = t[0].sum() #统计总字数
rt = [] #保存结果用

for m in range(2, max_sep+1):
    print(u'正在生成%s字词...'%m)
    t.append([])
    for i in range(m): #生成所有可能的m字词
        t[m-1] = t[m-1] + re.findall(myre[m], s[i:])

    t[m-1] = pd.Series(t[m-1]).value_counts() #逐词统计
    t[m-1] = t[m-1][t[m-1] > min_count] #最小次数筛选
    tt = t[m-1][:]
    for k in range(m-1):
        qq = np.array(list(map(lambda ms: tsum*t[m-1][ms]/t[m-2-k][ms[:m-1-k]]/t[k][ms
        tt = tt[qq]
    rt.append(tt.index)

def cal_S(sl): #信息熵计算函数
    return -((sl/sl.sum()).apply(log)*sl/sl.sum()).sum()

for i in range(2, max_sep+1):
    print(u'正在进行%s字词的最大熵筛选(%s)...'%(i, len(rt[i-2])))
    pp = [] #保存所有的左右邻结果
    for j in range(i+2):
        pp = pp + re.findall('(.)%s(.)'%myre[i], s[j:])
    pp = pd.DataFrame(pp).set_index(1).sort_index() #先排序，这个很重要，可以加快检索速度
    index = np.sort(np.intersect1d(rt[i-2], pp.index)) #作交集
    #下面两句分别是左邻和右邻信息熵筛选
    index = index[np.array(list(map(lambda s: cal_S(pd.Series(pp[0][s]).value_counts()
    rt[i-2] = index[np.array(list(map(lambda s: cal_S(pd.Series(pp[2][s]).value_counts
```

```
#下面都是输出前处理
for i in range(len(rt)):
    t[i+1] = t[i+1][rt[i]]
    t[i+1].sort(ascending = False)

#保存结果并输出
pd.DataFrame(pd.concat(t[1:])).to_csv('result.txt', header = False)
```

转载到请包括本文地址：https://kexue.fm/archives/3491

更详细的转载事宜请参考：《科学空间FAQ》

如果您需要引用本文，请参考：

苏剑林. (2015, Oct 26). 《新词发现的信息熵方法与实现 》[Blog post]. Retrieved from https://kexue.fm/archives/3491