

# 针对微博不实信息的获取与分析



作者 陈哲安 王源毅

学院 理学院

专业 数学与应用数学

学号 2018212432 2018212466

班级 2018214101

本文的代码由于markdown编辑器宽度受限，导致太长的代码换行的时候看起来有些问题 

# 摘要

针对微博社区管理中心的不实言论，我们调用了selenium进行数据的爬取，爬取包含不实信息和发帖人相关信息在内17项内容，共9985条数据。利用改进的杰卡德系数和余弦相似度进行了去重，最终得到不同谣言信息共1676条。

针对每条数据，我们设计了包含发帖人信用等级、性别、微博正文单词数在内的9项特征值，同时使用主成分分析法将数据特征降维至二维进行分析。在使用K-means聚类分析发现效果不好后，采用了高斯混合模型（GMM）进行聚类，最终表现效果良好，数据有明显的聚类趋势且聚类准确，得到5类数据。我们选取了其中“青少年死亡”一类有关的数据，按照时间将数据划分进行了时间序列分析，对原始数据进行一阶差分和ADF检验后，构建了ARIMA模型。模型拟合效果良好，正确率达87%，并利用该模型对这一谣言进行了预测，

我们还对每条谣言构建了词袋矩阵。在词袋矩阵的基础上使用TF-IDF进行了归一化操作，并训练出LDA主题模型，得到“疫情”、“青少年”等11个主题，并调用gensim和matplotlib库进行了数据可视化分析。

最后我们利用正常博文与谣言训练了支持向量机，用于判定博文是否为谣言，模型得分达到79%。

关键词：爬虫；TF-IDF；主成分分析；LDA；GMM；支持向量机

# 不实信息爬虫构建

---

前言：

最终经过讨论，我们决定爬取17项内容：

---

以下是我们确定的需要爬取的内容

---

举报人

---

性别 (举)

---

所在地区 (举)

---

被举报人

---

性别(被)

---

所在地区(被)

---

被举报人的微博主页

---

被举报内容的微博网址

---

被举报人信用等级

---

被举报人是否是微博会员

---

不实信息内有无超链接

---

不实信息具体内容

---

该举报在不实信息网站原文链接

---

站方判定

---

站方判定(全)

---

被举报微博发布时间

---

被举报微博发布时间 (精确)

---

用到的**modules**:

```
from selenium import webdriver
from pyquery import PyQuery as pq
import csv
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import time
from multiprocessing import Process
```

首先设置一个返回一个带头模式的浏览器的函数start\_webdriver()

```
def start_webdriver():
    # 登录浏览器
    options = webdriver.FirefoxOptions()
    # options.add_argument('--headless')
    # options.add_argument('--no-sandbox')
    # options.add_argument('--disable-dev-shm-usage')
    driver
    =webdriver.Firefox(options=options,executable_path='/usr/local/bin/geckodriver')
    return driver
```

然后利用start\_webdriver()函数先启动一个浏览器,获取cookies信息,以便后面多线程里的浏览器使用

```
# 手动到微博的登录界面然后登录
driver=start_webdriver()
driver.get('https://weibo.com/login.php')
#手动登录完毕之后获取 cookies
cookies=driver.get_cookies()
driver.close()
```

**data\_explorer(driver,href):**

让driver跳转到href对应的界面后,运用我们对于不实信息网页分析得到的各个我们所需信息的xpath,进行相应的爬取.对于每条不实信息,我们总共需要获取以下内容:

```
[ "举报人",
  "性别(举)",
  "所在地区(举)",
  "被举报人",
  "性别(被)",
  "所在地区(被)",
  "被举报人的微博主页",
  "被举报内容的微博网址",
```

```
        "被举报人信用等级",
        "被举报人是否是微博会员",
        "不实信息内有无超链接",
        "不实信息具体内容",
        "该举报在不实信息网站原文链接",
        "站方判定",
        "站方判定(全)",
        "被举报微博发布时间",
        "被举报微博发布时间(精确)",
    ]
```

data\_explorer(driver,href)函数运行的思路是:

- 首先寻找页面右边被举报人的区域里面是否有"阅读全文"的标签, 如果有的话, 就把阅读原文的标签全部点击

```
Yuanwen_flag=0
try:
    #把 右边的 阅读全文 全部点了
    t=driver.find_elements_by_xpath("//div[@class='part_report part_report_alone
clearfix']//div[@class='W_main_half_r']//a[@suda-
uatracc='key=tblog_service_account&value=view_original_text']")
    for i in t:
        i.click()
    if(t):
        Yuanwen_flag=1
        # print("有 原文 按钮")
    else:
        Yuanwen_flag=0
        # print("无 原文 按钮")
except:
    Yuanwen_flag=0
    # print("无 原文 按钮")
```

- 然后, 根据各个xpath对于网页进行扫描, 获取需要的信息

```
if(Yuanwen_flag==1): #有      点全文      按钮
    #被举报的信息
    try:
        t=driver.find_element_by_xpath("//div[@class='W_layer']//div[@class='layer_feed
feed clearfix']//div[@class='con']")
        # if(t.find('#微博辟谣')):
        #     return
        ## 不实微博文本有无超链接
        if t.text.find('http:')==-1 and t.text.find('https:')==-1:
```

```

        message_link_whether=0

    else:
        message_link_whether=1
        people2_wrong_message=t.text[t.text.find(': ')+1: ].split('http:')
[0].split('https:')[0]

    except:
        Yuanwen_flag=0
        if(Yuanwen_flag==0): #无 点击全文 按钮
            #被举报的信息
            t=driver.find_element_by_xpath("//div[@class='part_report part_report_alone
clearfix']//div[@class='W_main_half_r']//div[@class='feed_bg_orange2
clearfix']//div[@class='con']")
            ## 不实微博文本有无超链接
            if t.text.find('http:')==-1 and t.text.find('https:')==-1:
                message_link_whether=0
            else:
                message_link_whether=1
                people2_wrong_message=t.text[t.text.find(': ')+1: ].split('http:')
[0].split('https:')[0]

#####
#####
#举报人的性别
people1_sex=driver.find_element_by_xpath("//div[@class='part_report
clearfix']//div[@class='W_main_half_l']//p[@class='mb']//img").get_attribute("title
")
#举报人网名
people1_name=driver.find_element_by_xpath("//div[@class='part_report
clearfix']//div[@class='W_main_half_l']//div[@node-
type='report_user_area']//p[@class='mb W_f14']//a").text
#举报人所在地区
people1_area=driver.find_element_by_xpath("//div[@class='part_report
clearfix']//div[@class='W_main_half_l']//p[@class='mb']").text
#被举报人性别
people2_sex=driver.find_element_by_xpath("//div[@class='part_report
clearfix']//div[@class='W_main_half_r']//p[@class='mb']//img").get_attribute("title
")
#被举报人网名
people2_name=driver.find_element_by_xpath("//div[@class='part_report
clearfix']//div[@class='W_main_half_r']//div[@class='user_bg_orange2
clearfix']//p[@class='mb W_f14']//a").text
#被举报人所在地区
people2_area=driver.find_element_by_xpath("//div[@class='part_report
clearfix']//div[@class='W_main_half_r']//p[@class='mb']").text

# print('1')

temp=driver.find_elements_by_xpath("//div[@class='part_report
clearfix']//div[@class='W_main_half_r']//p[@class='mb W_f14']//a")

```

```

#被举报人微博
people2_weibo=temp[0].get_attribute('href')
#被举报人的信用等级
t=driver.find_element_by_xpath("//div[@class='part_report_fix']//div[@class='W_main_half_r']//p[@class='mb_W_f14']//img").get_attribute('title')
people2_credit_rating=t.split(':')[1][1]

# print('2')
#被举报信息原文链接
try:

people2_wrong_message_weibo_link=driver.find_element_by_xpath("//div[@class='part_report_fix']//div[@class='W_main_half_r']//p[@class='publisher']//a").get_attribute('href')
# people2_delete_or_not='否'
except:
people2_wrong_message_weibo_link="###"
# people2_delete_or_not='是'

#不实信息发布时间
t=driver.find_element_by_xpath("//div[@class='part_report_fix']//div[@class='W_main_half_r']//p[@class='publisher']")
#发布的精确时间
judge_time_precisely=t.text[t.text.find(':')+1:].split('|')[0]
if judge_time_precisely=="被举报微博":
    judge_time_precisely=""
#发布时间
judge_time=judge_time_precisely.split(" ")[0]
if judge_time=="被举报微博":
    judge_time=""

j=driver.find_element_by_xpath("//div[@class='middle_middle_long']//p").text
#站方判定(大概)
official_judgement=j[j.find('')+1:].split('')[0]
#站方判定(具体)
official_judgement_all=j

#该举报在不实信息网站原文链接
judge_href=href

# print('3')

#被举报人是否是会员
try:
    try:
        driver.get(people2_weibo)

```

```

        except:
            driver.execute_script('window.stop ? window.stop() :
document.execCommand("Stop");')
            # switch_to_active(driver)
            driver.find_element_by_xpath("//a[@title='微博会员']")
            people2_whether_member=1

        except:
            people2_whether_member=0
        # driver.get(href)
        # switch_to_active(driver)
    # print('4')

```

- 将这些信息汇总成一个列表，返回

```

per_data=[

    people1_name,
    people1_sex,
    people1_area,
    people2_name,
    people2_sex,
    people2_area,
    people2_weibo,
    people2_wrong_message_weibo_link,
    people2_credit_rating,
    people2_whether_member,
    message_link_whether,
    people2_wrong_message,
    judge_href,
    official_judgement,
    official_judgement_all,
    judge_time,
    judge_time_precisely,
    # people2_delete_or_not
]

# print(per_data)
print("该条信息已经爬完")
return per_data

```

### **data\_get(driver,href):**

上面的**data\_explorer**的外层函数。由于可能会出现页面一直在加载，但实际上页面的整体内容已经加载出来了的情况，最终会导致浏览器的超时。当浏览器跳转到对应的界面之后，一直如果还在加载并抛出异常的话，就执行相应的js语句，强行停止加载。

```

def data_get(driver,href):
    # tag.click()
    try:
        driver.get(href)
    except:
        driver.execute_script('window.stop ? window.stop() :
document.execCommand("Stop");')
    try:
        return data_explorer(driver,href)
    except:
        print("该条信息爬取失败")
        return

```

### main\_func:

最终由于cpu资源充足，我们选择利用多进程进行爬取，而main\_func就是我们提交给每个进程需要执行的任务。里面涉及到了：

- 对于每个进程的监控：
  - 利用打印的信息，判断进程是否在运行
  - 利用对每个进程设置的日志文件（对应count\_txt），判断当前已爬取的页数，以及爬取的每页的数据量
  - 利用日志文件里面开头和结尾的时间信息，判断每个进程最终运行的时间
- 对于爬取的每页数据（一个包含一页所有不实信息条目的href的列表），遍历每一个不实信息条目，运行data\_explorer(driver,href)
- 对于每页数据的写入：
  - 由于计算机的 I/O 操作需要占用大量的cpu时间，所以我们选择在爬完一整页之后在进行数据的写入
  - 目标500页都爬取完成之后，跳转到一个特殊网页作为标志

```

def main_func(start,need,date,number,cookies,home_page):
    driver=start_webdriver()
    driver.get('https://weibo.com/login.php')
    for i in cookies:
        driver.add_cookie(cookie_dict=i)
    print("第{}号浏览器开启成功".format(number))

    driver.set_page_load_timeout(10)

    # time.sleep(120)
    # for i in cookies:
    #     driver.add_cookie(i)
    count_txt="count_"+date+"_"+number+".txt"
    with open(count_txt, "a+", newline="") as g:
        writer1 = csv.writer(g)
        writer1.writerow(["第{}号进程正式开始于: {}".format(number,time.ctime())])

```

```

print("第{}号进程正式开始于: {}".format(number,time.ctime()))
g.close()

# driver.get('https://www.cnblogs.com/the-only-flash/')

with open("weibowrongMessages_"+date+"_"+number+".csv", "a+", newline="") as f:
    writer = csv.writer(f)
    # 确定要爬取什么东西
    writer.writerow([
        "举报人",
        "性别(举)",
        "所在地区(举)",
        "被举报人",
        "性别(被)",
        "所在地区(被)",
        "被举报人的微博主页",
        "被举报内容的微博网址",
        "被举报人信用等级",
        "被举报人是否是微博会员",
        "不实信息内有无超链接",
        "不实信息具体内容",
        "该举报在不实信息网站原文链接",
        "站方判定",
        "站方判定(全)",
        "被举报微博发布时间",
        "被举报微博发布时间(精确)",
    ])
    f.close()

#用循环实现爬取
for num in range(start,need+1):
    page_datas=[]
    print("现在开始爬取第{}页".format(num))
    try:
        driver.get( home_page+str(num) )
    except:
        driver.execute_script('window.stop ? window.stop() : document.execCommand("Stop");')
    #获取该列表页所有不实信息标签

    all_tag=driver.find_elements_by_xpath('//div[@class="m_table_tit"]//a[@target="_blank"]')
    haven_count=0

    all_tag_href=[]
    for tag in all_tag:
        all_tag_href.append(tag.get_attribute('href'))

    for href in all_tag_href:
        # driver.get(href)
        haven_count=haven_count+1
        print("第{}页 第{}条 数据".format(num,haven_count))

```

```

#获取页面数据
data_per=data_get(driver,href)
if(not data_per):
    print("ERROR")
page_datas.append(data_get(driver,href))

#一页的数据压进 csv
with open("weiboWrongMessages_"+date+"_"+number+".csv", "a+", newline="")
as w:
    writer = csv.writer(w)
    counted=0
    for i in page_datas:
        if(not i):
            continue
        # print(i)
        counted=counted+1
        writer.writerow(i)
    print("第{}页一共存了{}条数据@@@@@@@".format(num,counted))
    w.close()

#每获取完一页的数据时候写一次, 因为代码的准确性已经可以确定, 现在要最求快速
with open(count_txt, "a+", newline="") as g:
    writer1 = csv.writer(g)
    writer1.writerow(["第{}页已爬完##### 一共存了{}条数据
@@@@@@@".format(num,counted)])
    g.close()

driver.get('https://www.cnblogs.com/the-only-flash/')
with open(count_txt, "a+", newline="") as k:
    writer1 = csv.writer(k)
    writer1.writerow(["第{}号进程最终结束于: {}".format(number,time.ctime())])
    print("第{}号进程最终结束于: {}".format(number,time.ctime()))
    k.close()

```

## 创建多进程:

查阅自己电脑的信息, 发现一共有4个逻辑内核, 最终选择利用4个进程

```

#start,need,date,number
processes = []
target_pages=500
num=4
average=int(target_pages/num)
date="5_9"
home_page="https://service.account.weibo.com/index?
type=5&status=4&page="#+str(page)
for i in range(num):
    processes.append( Process(target=main_func,args=(average*i+1,average*
(i+1),date,str(i+1),cookies,home_page,)) )

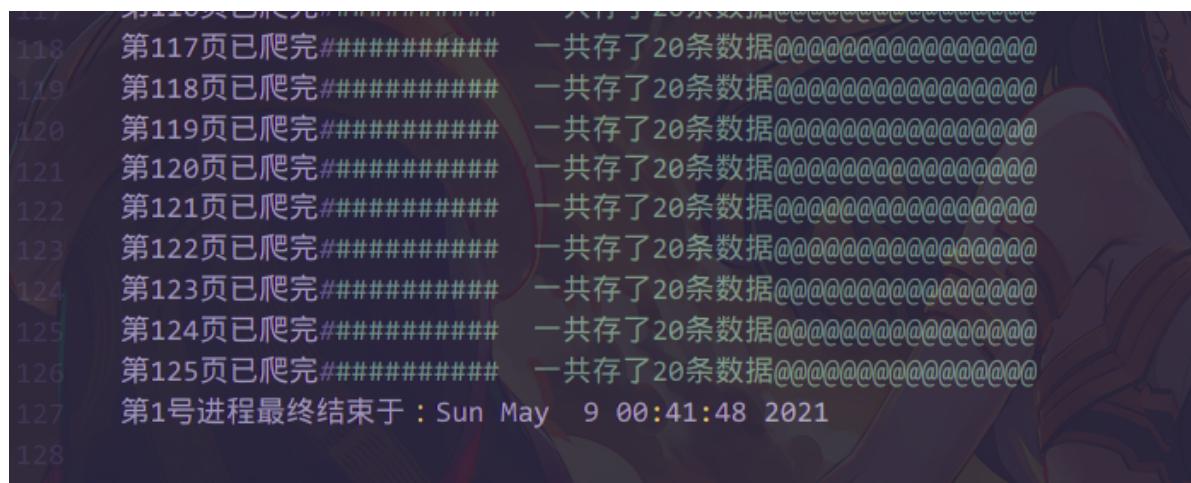
```

小结：

四个进程最终平均运行时间大概为7个小时

所有进程一共爬取了9985条不实信息

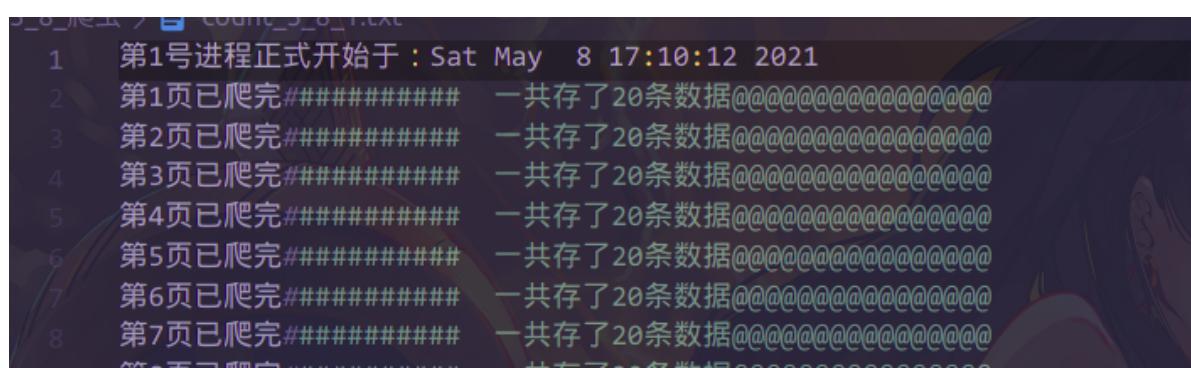
下面是5月9号第一次爬取测试中第一号进程的运行时间(实际上是5月10号爬的，但是日志删掉了②)：



```

117 第116页已爬完#####
118 第117页已爬完#####
119 第118页已爬完#####
120 第119页已爬完#####
121 第120页已爬完#####
122 第121页已爬完#####
123 第122页已爬完#####
124 第123页已爬完#####
125 第124页已爬完#####
126 第125页已爬完#####
127 第1号进程最终结束于 : Sun May 9 00:41:48 2021
128

```



```

1 第1号进程正式开始于 : Sat May 8 17:10:12 2021
2 第1页已爬完#####
3 第2页已爬完#####
4 第3页已爬完#####
5 第4页已爬完#####
6 第5页已爬完#####
7 第6页已爬完#####
8 第7页已爬完#####

```

最后我们得到了包含9985条不实信息数据的“all\_5\_10.csv”

结果截图：

	column 4	column 5	column 6	column 7	column 8
1	被举报人	性别(被)	所在地区(被)	被举报人的微博主页	被举报内容的微博网
2	乔木DC	男	海外 美国	http://weibo.com/u/6914968045	http://weibo.com/69
3	贵州榜哥	男	贵州	http://weibo.com/u/5392590366	http://weibo.com/53
4	小七幽默	女	广东 深圳	http://weibo.com/u/7582508763	http://weibo.com/75
5	小多多emm	男	北京	http://weibo.com/u/1918083051	###
6	吃甜皮鸭的土拨鼠	女	湖南 长沙	http://weibo.com/u/6050451455	###
7	老黄论天下	男	广东 广州	http://weibo.com/u/3721723284	###
8	飞鹏319	男	其他	http://weibo.com/u/1564617371	http://weibo.com/15
9	李不白的微博	男	其他	http://weibo.com/u/1901225922	http://weibo.com/19
10	蠢猪的核弹已命中	男	其他	http://weibo.com/u/3807663839	http://weibo.com/38
11	蠢猪的核弹已命中	男	其他	http://weibo.com/u/3807663839	http://weibo.com/38
12	蠢猪的核弹已命中	男	其他	http://weibo.com/u/3807663839	###
13	精分总监	男	江苏	http://weibo.com/u/5033795713	###
14	我爱我的国528我爱我的家	男	香港	http://weibo.com/u/7265799736	###
15	中国正能量	男	山东	http://weibo.com/u/1841515721	http://weibo.com/18
16	每日一善工作室	男	云南	http://weibo.com/u/7010546104	###
17	中原-老张	男	河南 郑州	http://weibo.com/u/7509646499	###
18	帝哥说事儿	男	辽宁	http://weibo.com/u/2155226773	###
19	暴哥猛料	女	北京	http://weibo.com/u/5074648251	###
20	中国正能量	男	山东	http://weibo.com/u/1841515721	###
21	人民正能量	女	北京	http://weibo.com/u/2307318984	###
22	叨叨那些事鸭	女	上海	http://weibo.com/u/7565729612	http://weibo.com/75

## 数据处理

### 数据清洗

用到的包：

```
import pprint
from collections import Counter
import jieba
import numpy as np
import pandas as pd
import csv
import os
import jieba.posseg as pseg
import sys
import string
from sklearn import feature_extraction
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from gensim import similarities
import jieba.analyse
import pprint
from collections import Counter
import jieba
import numpy as np
import pandas as pd
import csv
import os
import jieba.posseg as pseg
import sys
import string
from sklearn import feature_extraction
from gensim import similarities
```

```
import jieba.analyse  
import matplotlib.pyplot as plt
```

## 1.去重

- 针对爬取到的每条数据，都含有站方判定与被举报微博发布时间两项特征，我们先对被举报微博发布时间进行处理：由于爬取到的时间这一特征是具体年月日形式，我们将其按照月为划分，形成一项新的特征——时间戳。

	column 16	column 17	column 18
0分钟内生效。	被举报微博发布时间	被举报微博发布时间(精确)	时间戳
	2021-05-01	2021-05-01 09:01:51	202105
	2021-05-08	2021-05-08 09:45:55	202105
	2021-05-07	2021-05-07 11:02:48	202105
	2021-05-02	2021-05-02 17:03:36	202105
	2021-05-02	2021-05-02 13:30:35	202105
	2021-05-09	2021-05-09 12:30:52	202105
	2021-04-12	2021-04-12 19:00:36	202104
	2021-04-12	2021-04-12 20:53:51	202104
	2021-04-09	2021-04-09 00:36:30	202104
	2021-04-01	2021-04-01 12:25:30	202104

- 由于站方判定中，微博平台已经将相同的谣言用同一叙述方式总结出，如：博主A发布谣言xxxxabc；博主B发布谣言xxxedf。这两者为同一谣言内容，在微博社区管理中心板块，大部分不实信息页面中央的站方判定内容中，自动将其归为

经查，该微博称“×××”

例如：



经查，该微博称“山东省潍坊市国际风筝节时，因为家长没有及时照看好孩子，造成风筝尾巴将一小孩缠住带上了天空”不实  
我们先根据上一步设置的时间戳特征，在相邻两个时间戳：即本月、下月两个时间内，进行站方判定内容的相似度对比，相同的或相似度高的数据删除，不同数据保存，每条谣言只保留最早一条数。

- 我们爬取的结果证明相似微博的站方判定内容“”里面的内容基本一致

Ji7a4c	晒晒河北省曲周县依庄乡多个村庄人居住环境
Ji7a4g	晒晒河北省曲周县依庄乡多个村庄人居住环境
Ji7a0g	晒晒河北省曲周县依庄乡多个村庄人居住环境
Ji7a0h	晒晒河北省曲周县依庄乡多个村庄人居住环境
Ji7a0i	晒晒河北省曲周县依庄乡多个村庄人居住环境
Ji7a0j	晒晒河北省曲周县依庄乡多个村庄人居住环境
Ji7ask	晒晒河北省曲周县依庄乡多个村庄人居住环境
Ji7asl	晒晒河北省曲周县依庄乡多个村庄人居住环境

- （伪杰卡德距离）观察我们爬取的数据，由于相似的微博内容中，较短的微博往往更具代表性，并且前半部分（重合部分）往往一致，所以我们选取次数较少的那组数据的模作为分母，在杰卡德距离的基础上做了细微的调整，当然最后必须要保证小于1这个特性。——当然了，不可避免的会出现误删的现象

特朗普带着夫人去探望奥巴马，并且送上礼物，企图平息黑人的怒火。但奥巴马接过礼物，就将其甩了出去 这是今天全球最疯传的视频 有种就不要伪装嘛！#国际资讯#超话#美国总统特朗普（也开始学中国走亲访友了）  
特朗普带着夫人去探望奥巴马，并且送上礼物，企图平息黑人一族的怒火。没想到奥巴马接过礼物，一转身便将其甩了出去[呲牙]  
网友投稿：在我家干了8年的保姆阿姨去世了，妻子哭的痛不欲生，在收拾阿姨遗物时，发现了150000现金、一对玉镯和一封信。我和妻子来郑州奋斗已经是第10年了，在我家干了8年的保姆阿姨去世了，妻子哭的痛不欲生，在收拾阿姨遗物时，发现了150000现金、一对玉镯和一封信。我和妻子来郑州奋斗已经是第10个年头，我为确保国际社区血统纯正，禁止中国人居住？[责解]  
特朗普带着夫人去探望奥巴马，并且送上礼物，企图平息黑人一族的怒火。没想到奥巴马接过礼物，一转身便将其甩了出去这是今天的视频，真解气！  
奥巴马当众扔了川普的礼物[喵喵][喵喵][喵喵][喵喵]#奥巴马就美国暴乱发声#  
特朗普带着夫人去探望奥巴马，并且送上礼物，企图平息黑人一族的怒火。没想到奥巴马接过礼物，一转身便将其甩了出去.....[捂脸][捂脸]  
#美国75岁男子被警察推倒受重伤#特朗普带着夫人去探望奥巴马，并且送上礼物，企图平息黑人一族的怒火。没想到奥巴马接过礼物，一转身便将其甩了出去[呲牙]  
特朗普带着夫人去探望奥巴马，并且送上礼物，企图平息黑人一族的怒火。没想到奥巴马接过礼物，一转身便将其甩了出去[呲牙] 这是今天的视频#美国暴乱##美国#  
#美国#带#着#夫#人#去#探#望#奥#巴#马#，#并#且#送#上#礼#物#，#企#图#平#息#黑#人#一#族#的#怒#火#。#没#有#想#到#奥#巴#马#接#过#礼#物#，#一#转#身#便#将#其#甩#了#出#去#这#是#今#天#全#球#最#疯#传# 的#视#频#

$$J_{\text{原}}(A, B) = \frac{|A \cap B|}{|A \cup B|} \rightarrow J_{\text{伪}}(A, B) = \frac{|A \cap B|}{\min(|A|, |B|)}$$

- 余弦相似度：

我们直接利用了网上关于余弦相似度的公式

$$L_{\text{余弦相似度}}(A, B) = \frac{A \cdot B}{|A||B|}$$

code:

```
#计算文本相似度 @ (伪) 杰卡德系数
def calculateSimilarity(s1, s2):# 计算s1在s2中的相似度
    def add_space(s):
        return ' '.join(cleanWord(str(s)))

    # 将字符串中间加入空格
    s1, s2 = add_space(s1), add_space(s2)

    # 转化为TF矩阵
    from sklearn.feature_extraction.text import CountVectorizer
    cv = CountVectorizer(tokenizer=lambda s: s.split())

    corpus = [s1, s2]
    if not s1 and not s2:
        return -3
    if not s1:
        return -1
    if not s2:
        return -2
    vectors = cv.fit_transform(corpus).toarray()#转成向量并且放入数组中
    # 求交集
    numerator = 0
    for i in np.min(vectors, axis=0):
        if i != 0:
            numerator+=1
    # 求并集(伪)
    a1=0
    a2=0
```

## 小结：

通过以上

### [1]有关站方判定的选取

[2]规定每条数据在本月和下月中进行去重

### [3]两个距离的选取

我们分别进行了4步操作：

(以下所有对于距离去重的内容，我们统一使用阈值0.50000)

- 首先对站方判定中的精确内容利用伪杰卡德距离去重
- 接着，做了一些细节处理



```
# 做一些细节处理
data=pd.read_csv('5_10_corrected.csv',sep=',')
kepted_rows=list(range(0,data.shape[0]))
for i in range(data.shape[0]):
    if "经查" not in str(data.iloc[i][14]):
        kepted_rows.remove(i)
data=data.iloc[kepted_rows]
data.to_csv('5_10_corrected_1.csv',index= 0)
```

- 其次在利用余弦相似度对于站方判定中的精确内容进行了去重
- 最后对于不实信息具体内容，我们又进行了利用余弦相似的去重

最终，我们生成了经过多次去重后的文件“5\_10\_corrected\_3.csv”

里面包含1676条数据

## 2.词袋矩阵以及字典的构建

前言：

一开始，我们直接利用 `jieba` 的分词功能获得了一个字典，大小为9926.

但是，对于这次大作业的词袋矩阵进行构建时候，有一个要求是取出不常用词，这里直观的想法是使用TF（词频），通过设定阈值来判断是否为常用词，但是如果单凭词频来区分常用与不常用的词语的话，最后可能会删掉一些词词频不高，但是却十分重要的词语，转而留下一些词频很高但是不重要的词语，比如一些数字还有量词什么的（这些才应该是要删掉的）

例如：

123	一两分钟	19
124	一个三十多	19
125	一个三岁	19
126	一个四十多	19
127	一个多	19
128	一个打	19

	名称框	B	
1	100	214	
2	12	176	
3	13	93	
4	17	90	
5	170cm	80	
6	20km	80	
7	21d	69	

所以我们没有使用利用词频的方式来判定（是/否）常用词，而是利用了结巴自带的 `jieba.analyse.extract_tags (top_k)`：函数传入的参数是一篇文档，然后计算这个这个文档中所有词语的  $TF(word) \times IDF^*(word)$ ，其中  $IDF^*$  是结巴利用自己的语料库训练得到的IDF值，返回前 `top_k` 个关键词

下面是CSDN博客：[jieba关键词提取的源码解析](#)中的部分内容

```

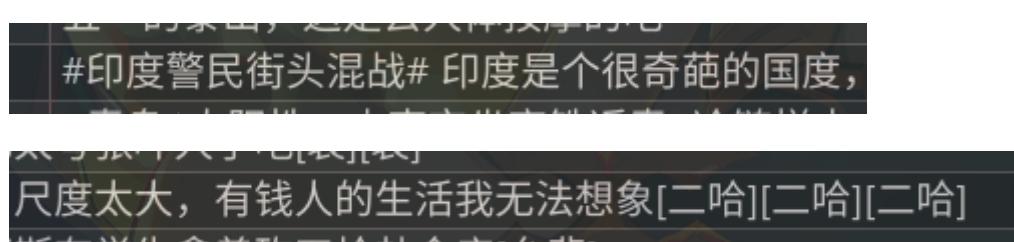
43 kw = k.word if allowPOS and withFlag else k
44 # 依据tf-idf公式进行tf-idf值，作为词的权重。其中，idf是jieba通过语料库统计得到的
45 freq[k] *= self.idf_freq.get(kw, self.median_idf) / total
46

```

实际中我们是把切分过的语料库合成了一篇文档（空格分割每个词语），然后返回了前2000个关键词作为我们选定的常用词（关键词？）

不实信息文本的进一步清洗：

我们组在爬取数据的时候，默认返回了不带有超链接（对应[标签](#)，内容为“<http://>”或“<https://>”）的数据，但是由于里面存在很多的表情 `[xxx]` 还有[hashtag](#) `#xxx#`,



#印度警民街头混战# 印度是个很奇葩的国度，  
尺度太大，有钱人的生活我无法想象[二哈][二哈][二哈]

我们为了获取到不包含这些信息的文本内容，又利用[re](#)库的正则表达式进行了进一步的处理

```

import re
import pandas as pd
data=pd.read_csv("./5_10_corrected_3.csv")
data2=data["不实信息具体内容"]

```

```

pattern = re.compile(r"#+.*?#+")
pattern1 = re.compile(r"\[.*?\]",re.S)
with open("alterContent_1.csv",'a+') as f:
    writer=csv.writer(f)
    for i in range(1,len(data2)):
        ans=str(data2[i]).split(": ")[1:]
        print(ans)
        ans="".join(ans)
        m=re.sub(pattern,"",ans)
        m=re.sub(pattern1,"",m)
        writer.writerow([m])
f.close()

```

经过上面的操作，最终我们获取了“alterContent\_1.csv”

停用词表的获取：

在得到词袋矩阵的过程中，我们需要利用停用词表来删除一些在语法中表示停顿的词语和符号，比如：‘不时’，‘～’，‘总之’ ...

在本次实验中，我们选择了将以下停用词表（来自github）以及将我们认为属于停用词的一些词语进行整合

得到了一个新的停用词表文件“worddd.txt”

## 中文常用停用词表

词表名	词表文件
中文停用词表	cn_stopwords.txt
哈工大停用词表	hit_stopwords.txt
百度停用词表	baidu_stopwords.txt
四川大学机器智能实验室停用词库	scu_stopwords.txt
请吃辣条	

## 利用上述操作获得词袋矩阵以及字典

利用前面所讲的内容，我们构造如下的**transform\_1()**函数，返回的是一个词袋矩阵以及一个字典

```
def transform_1(file_path):
    """
    1.transform_1()和transform()的区别在于利用jieba库选定了前2000个关键词，也就是词袋
    2.wordddd.txt是经过多次组合后的停用词表
    将原文本数据转化为 词向量矩阵 ndarray
    file_path: 原始的按行存储的文本数据 路径
    返回的是词向量矩阵还有，字典
    """

    content = []
    with open(file_path, 'r', encoding="utf-8") as f:
        content=f.readlines()
        f.close()
    #对文档进行分词处理，采用默认模式
    # seg_list_many=[]wrong_message_content.csv
    seg_list=[]
    for i in content:
        seg_list.append( list(jieba.cut(i,cut_all=False)) )
        # seg_list_many.append(seg_list)

    #运用stopwords对进行处理
    corpus=[]

    stopwords_raw=[]
    stopwords=[]
    with open("wordddd.txt", encoding="utf-8") as f:
        stopwords_raw = f.readlines()
    for i in stopwords_raw:
        stopwords.append( "".join(i.split("\n")[:-1]) )

    for seg in seg_list:
        tmp=[]
        for i in seg :
            # weight_list_tmp=[]
            if i not in stopwords:
                try:
                    float(i)
                except:
                    tmp.append(i)
        tmp = ' '.join(tmp)
        corpus.append(tmp)

    content_word=" ".join(corpus)
    tags = jieba.analyse.extract_tags(content_word, topK=2000)

    weight_list=[]
```

```
for seg in seg_list:
    weight_list_tmp=[0]*len(tags)
    for j in seg:
        for i in range(len(tags)):
            if tags[i] == j:
                weight_list_tmp[i]+=1
            else:
                pass
    weight_list.append(weight_list_tmp)

weight=np.array(weight_list)

return weight,tags
```

## 小结：

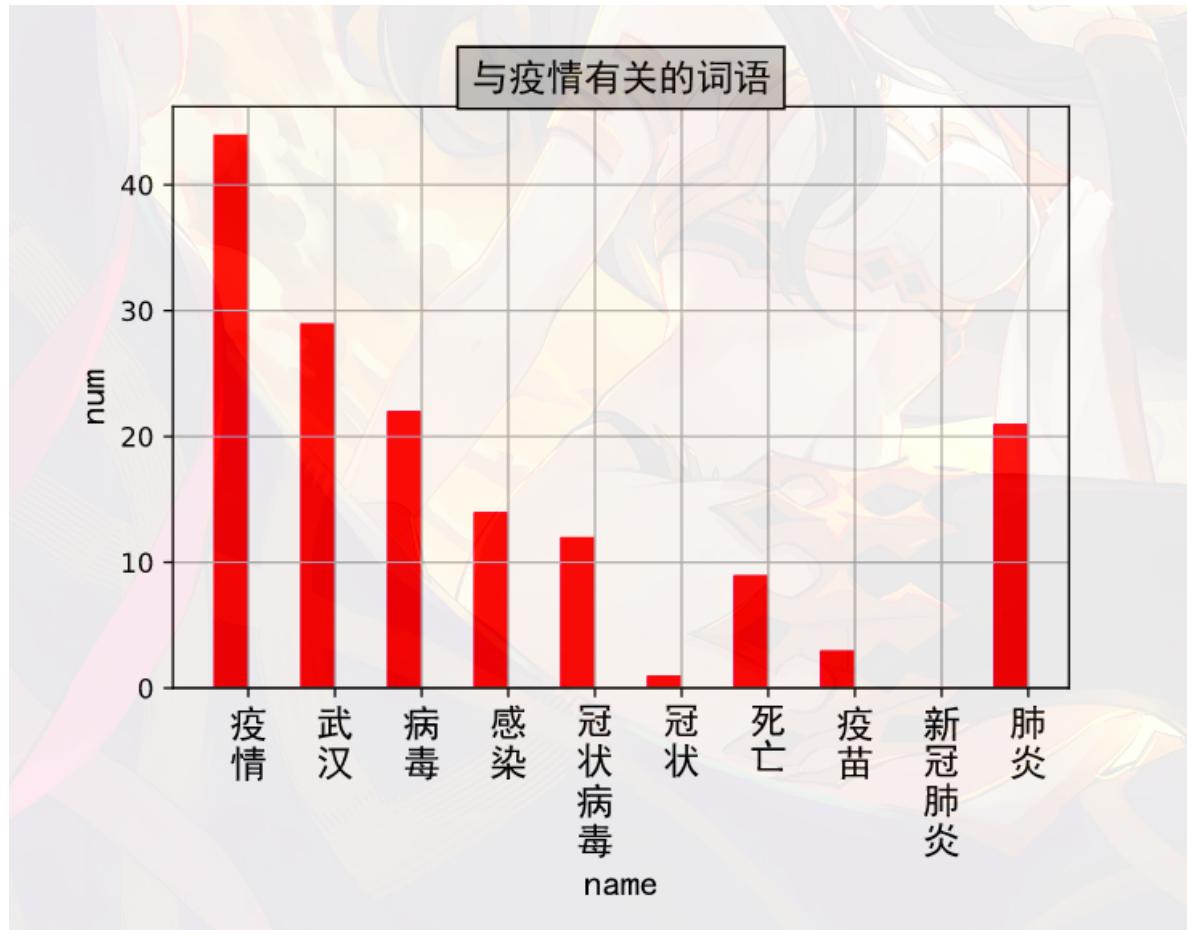
最终，我们运行了如下语句

```
a,b=transform_1('alterContent_1.csv')
```

### 3.字典中词语的频次展示（词云）



## 4.2020年之后与疫情相关词语的频次展示



### ④TF-IDF矩阵的构建以及pca降维

在“数据清洗”过程中，我们构建出了包含2000个重要词汇的词袋和对应微博正文的矩阵。我们利用TF-IDF对词袋归一化，算出了所有词汇的TF-IDF得分，原理以及实现过程如下。

TF-IDF是一种统计方法，用以评估一字词对于一个文件集或一个语料库中的其中一份文件的重要程度。字词的重要性随着它在文件中出现的次数成正比增加，但同时会随着它在语料库中出现的频率成反比下降。

其中，词频 (TF) 指的是某一个给定的词语在该文件中出现的次数。这个数字通常会被归一化(一般是词频除以文章总词数)，以防止它偏向长的文件。（同一个词语在长文件里可能会比短文件有更高的词频，而不管该词语重要与否。）公式为：

逆向文件频率 (IDF) 的主要思想是：如果包含词条t的文档越少，IDF越大，则说明词条具有很好的类别区分能力。某一特定词语的IDF，可以由总文件数目除以包含该词语之文件的数目，再将得到的商取对数得到。

某一特定文件内的高词语频率，以及该词语在整个文件集合中的低文件频率，可以产生出高权重的TF-IDF。因此，TF-IDF倾向于过滤掉常见的词语，保留重要的词语。

导入包：

```
import pprint
from collections import Counter
import jieba
import numpy as np
import pandas as pd
import csv
import os
import jieba.posseg as pseg
import sys
import string
from sklearn import feature_extraction
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from gensim import similarities
import jieba.analyse
import matplotlib.pyplot as plt
```

计算tf-idf矩阵的函数：

```
def func_tf_idf(weight):
    """
    得到词向量矩阵，经过tf-idf方法映射后，得到的矩阵
    weight: 词向量矩阵 weight=vectorizer.fit_transform(corpus)
    """
    tf_idf_transformer = TfidfTransformer()
    tfidf = tf_idf_transformer.fit_transform(weight)
    tfidf_weight = tfidf.toarray() #tf-idf矩阵 (实际)
    return tfidf_weight
```

我们将上文的词袋矩阵a进行tf-idf矩阵构造：

```
c=func_tf_idf(a)
```

结果截图：

可以看到结果是一个稀疏矩阵形式

	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0.403436...	0	0	0	0	0	0	0
2	0	0	0	0	0	0.233483...	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	0	0	0	0	0.433606...	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	0	0	0	0.066056...	0	0	0	0	0	0	0	0	0	0.157
20	0	0	0	0	0	0	0	0	0	0	0	0.310686...	0	0
21	0.202089...	0	0	0	0.492021...	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0

## tf-idf矩阵的pca降维

pca的函数如下：

```
def func_pca(weight,keeped_num):
    """
    返回的东西一看就懂
    """
    from sklearn.decomposition import PCA
    pca = PCA(n_components=keeped_num)
    pca.fit(weight)
    a=pca.explained_variance_ratio_
    b=pca.explained_variance_
    weight_new = pca.transform(weight)
    return weight_new,a,b
```

一开始先对tf-idf矩阵降到100维，但是观察他们的方差膨胀因子（ratio形式）：

0	0.009467...
1	0.007572...
2	0.006548...
3	0.006285...
4	0.005945...
5	0.005676...
6	0.005133...
7	0.004947...
8	0.004338...
9	0.004240...
10	0.004173...
11	0.004078...
12	0.004020...
13	0.003969...
14	0.003906...
15	0.003752...
16	0.003738...
17	0.003690...
18	0.003643...
19	0.003610...
20	0.003515...
21	0.003472...
22	0.003427...
23	0.003415...

我们看到连排到前几的方差膨胀因子占比都很小，所以基本不适合降维

如果要强行降到指定的维数  $k$  只需运行如下代码：

```
#weight是词袋矩阵  
weight_new,a,b=func_pca(weight,k)
```

## ④数据特征提取

根据上一步处理完的数据，我们设计了9项特征值来衡量某一微博正文，以便进行后续分析。这9个特征值是：

特征	说明
被举报人性别	0是女性，1代表男性
地区	1代表南方地区，2代表北方地区，0代表其他
信用等级	1代表正常，0代表中，-1代表差
是否会员	0代表否，1代表是
微博正文有无超链接	0代表否，1代表是
字符数	标点符号的个数（我们自己设置了一个标点集合构成的txt: punctuation_set.txt）
句子数	计算句子的数量(以句点，感叹号，问号分隔)
单词数	去除标点之后剩余的单词数（jieba分词）
平均句子长度	单词数/句子数

以前三条不实信息为例，它的9项特征值和其正文内容如下：

## 数据分析

### ⑤基于词袋矩阵的LDA模型

#### LDA模型的介绍：

--From 维基百科

##### 隐含狄利克雷分布 [\[编辑\]](#)

维基百科，自由的百科全书

隐含狄利克雷分布（英语：Latent Dirichlet allocation，简称LDA），是一种主题模型，它可以将文档集中每篇文档的主题按照概率分布的形式给出。同时它是一种无监督学习算法，在训练时不需要手工标注的训练集，需要的仅仅是文档集以及指定主题的数量k即可。此外LDA的另一个优点则是，对于每一个主题均可找出一些词语来描述它。

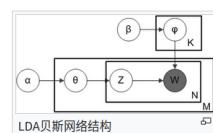
LDA首先由 David M. Blei、吴恩达和迈克尔·I·乔丹于2003年提出<sup>[1]</sup>，目前在文本挖掘领域包括文本主题识别、文本分类以及文本相似度计算方面都有应用。

## 数学模型 [编辑]

LDA是一种典型的词袋模型，即它认为一篇文档是由一组词构成的一个集合，词与词之间没有顺序以及先后的关系。一篇文档可以包含多个主题，文档中每一个词都由其中的一个主题生成。

另外，正如Beta分布是二项式分布的共轭先验概率分布，狄利克雷分布作为多项式分布的共轭先验概率分布。因此正如LDA贝斯网络结构中所描述的，在LDA模型中一篇文档生成的方式如下：

- 从狄利克雷分布 $\alpha$ 中取样生成文档*i*的主题分布 $\theta_i$
  - 从主题的多项式分布 $\theta_i$ 中取样生成文档*i*第*j*个词的主题 $z_{i,j}$
  - 从狄利克雷分布 $\beta$ 中取样生成主题 $z_{i,j}$ 的词语分布 $\phi_{z_{i,j}}$
  - 从词语的多项式分布 $\phi_{z_{i,j}}$ 中采样最终生成词语 $w_{i,j}$



因此整个模型中所有可见变量以及隐藏变量的联合分布是

$$p(w_i, z_i, \theta_i, \Phi | \alpha, \beta) = \prod_{j=1}^N p(\theta_i | \alpha) p(z_{i,j} | \theta_i) p(\Phi | \beta) p(w_{i,j} | \phi_{z_{i,j}})$$

最终一篇文档的单词分布的最大似然估计可以通过将上式的 $\theta_i$ 以及 $\Phi$ 进行积分和对 $z_i$ 进行求和得到

$$p(w_i|\alpha, \beta) = \int_{\theta_i} \int_{\Phi} \sum_{z_i} p(w_i, z_i, \theta_i, \Phi | \alpha, \beta)$$

根据 $p(w_i | \alpha, \beta)$ 的最大似然估计，最终可以通过吉布斯采样等方法估计出模型中的参数。

简单的来说，LDA就是以假设

- 每个文档 $d_i$ 主题分布是多项分布, 其先验分布是参数 $\alpha$ 的狄利克雷分布
  - 每个主题 $t_i$ 的词语分布是多项分布, 其先验分布是参数 $\beta$ 的狄利克雷分布

然后基于这个假设构造了一个语料库主题以及词语分布模型的概率密度函数

$$p(w_i, z_i, \theta_i, \Phi | \alpha, \beta) = \prod_{j=1}^n p(\theta_i | \alpha) p(z_{i,j} | \theta_i) p(\Phi | \beta) p(w_{i,j} | \phi_{z_{i,j}})$$

最后可以通过 [1]极大似然估计 [2]吉布斯采样 的方式对于模型的参数进行估计

而在实际运用中，我们一般用吉布斯采样这样一种迭代的方式进行LDA模型的构建，算法的简略描述如下：

- 随机对每个每个文档的每个词语分配一个主题，然后初始化一些变量的值
  - 然后遍历每个文档的每个主题，假设遍历到文档 $d_i$ 的词语 $w_j$ ，先假设它不属于任何主题，接着更新第一步提到的那些变量
  - 分别计算

	T1	T2	T3
Document-1	2	1	2-1=1
Document-m			

Where  $n_{ik}$  is the total number of words in  $i^{\text{th}}$  document in  $k^{\text{th}}$  topic,  $N_i$  is the number of words in the  $i^{\text{th}}$  document,  $K$  is the number of topics considered,  $\alpha$  is a hyper parameter

For instance, for document-1 and topic-3,  $p(t_k|d_i) = \frac{1+0.1}{5-1+3\times 0.1} = 0.155$  assuming  $\alpha = 0.1$ .

	T1	T2	T3
monuments	1	0	35-1 =34
restaurants	50	0	1
great	42	1	0
good	0	0	20
transport	10	8	1

Where  $m_{j,k}$  is the corpus wide assignment of word  $w_j$  to  $k^{\text{th}}$  topic.  $V$  is the vocabulary of the corpus.  $\beta$  is a hyper parameter

For instance, for the word monument in topic-3,  $p(w_j|t_k) = \frac{34+0.5}{56+5 \times 0.5} = 0.56$  assuming  $\beta = 0.5$

- 根据上面算出的值，然后对于文档 $d_i$ 的词语 $w_j$ ，分别对每个主题 $t_i$  ( $i = 1, \dots, k$ ) 计算

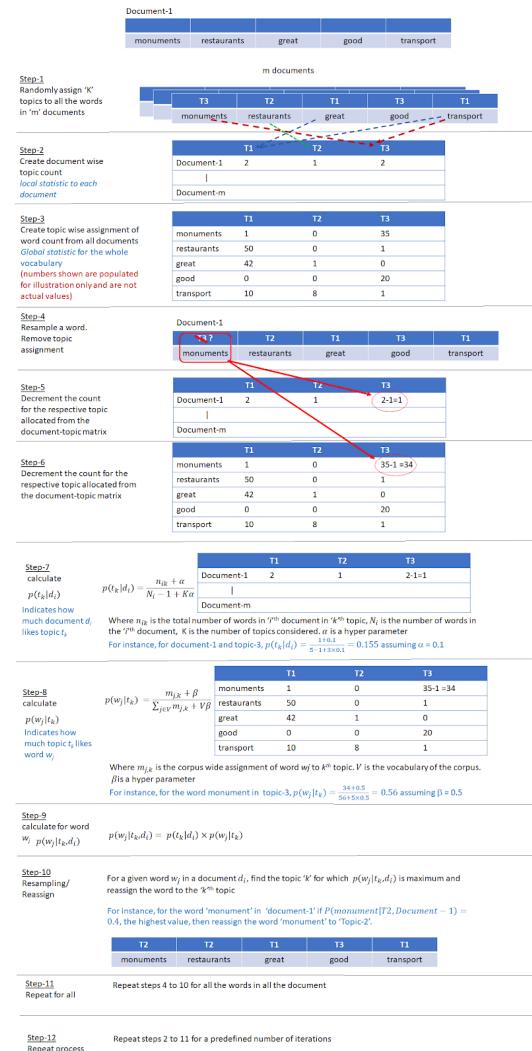
$$\text{calculate for word } w_j \quad p(w_j|t_k, d_i) \quad p(w_j|t_k, d_i) = p(t_k|d_i) \times p(w_j|t_k)$$

取最大值对应的那个 $t_i$ 作为该词语的主题

- 依次遍历所有文档的所有词语

这就构成了吉布斯采样的一次迭代

具体算法流程 (from 链接) :



## LDA实验部分：

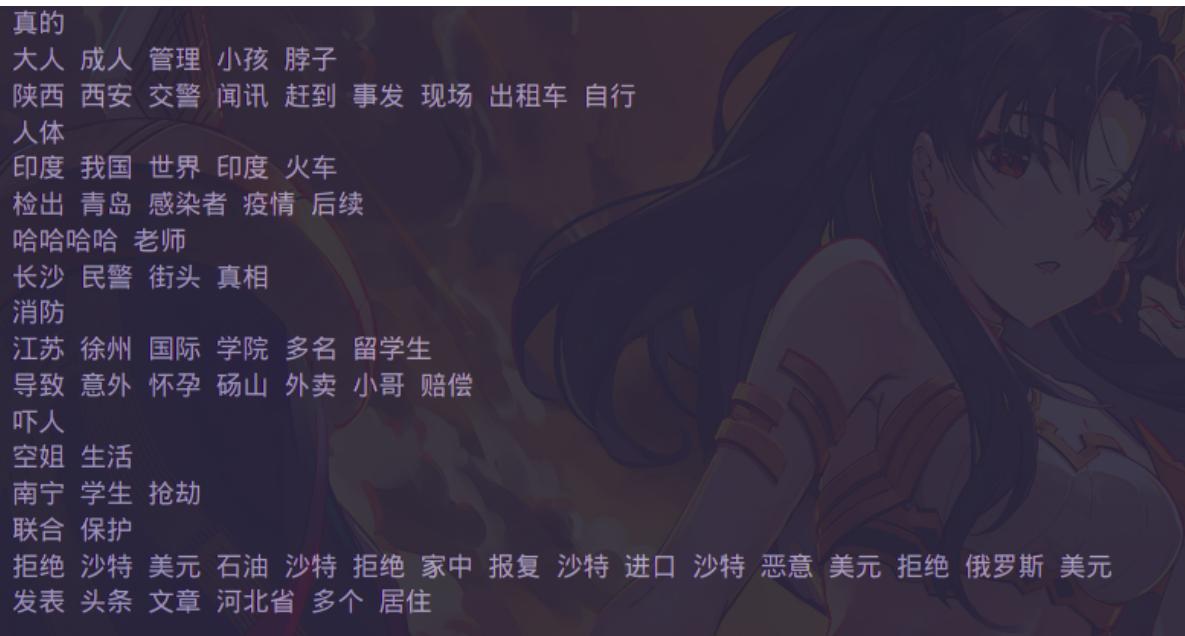
前言：

对于LDA主题模型的构建，我们用到了gensim库

主要利用了 `from gensim.models.ldamodel import LdaModel`

对于LDA模型的构建，以下的内容需要进行说明：

- 由于要用到gensim库的函数，我们相应的词袋矩阵以及字典的形式要符合gensim库的格式，所以这里词袋的构建与之前不同，gensim库要求每条文本被划分成如下形式（词语间以空格分开），我们利用前面的“`alterContent_1.csv`”利用停用词表，很容易得到如下所示的“`temp.txt`”



真的  
大人 成人 管理 小孩 脖子  
陕西 西安 交警 闻讯 赶到 事发 现场 出租车 自行  
人体  
印度 我国 世界 印度 火车  
检出 青岛 感染者 疫情 后续  
哈哈哈哈 老师  
长沙 民警 街头 真相  
消防  
江苏 徐州 国际 学院 多名 留学生  
导致 意外 怀孕 砀山 外卖 小哥 赔偿  
吓人  
空姐 生活  
南宁 学生 抢劫  
联合 保护  
拒绝 沙特 美元 石油 沙特 拒绝 家中 报复 沙特 进口 沙特 恶意 美元 拒绝 俄罗斯 美元  
发表 头条 文章 河北省 多个 居住

- 由于jieba库的分割函数可以返回词语的词性，我们只选出了词性为动词和名词的那些词语构成字典，并且字典的构造用到了类似前面构建词袋矩阵的时候**transform\_1()**函数返回的字典（写了个函数**getTags()**）放`lemmatization`函数的**tags**参数上

利用下面的`lemmatization`函数，我们得到了符合要求的一个列表corpus，corpus中的每个元素为上一条中提到的格式（以空格分隔）

```
def lemmatization(file_path, tags):  
    ...  
    保留属于"我们想要保留的词性"的词语  
    ...  
    # 这个是我们想要保留的词性  
    flags_we_wanted=[  
        'n' ,  
        'ng' ,  
        'nr' ,  
        'nrfg' ,  
        'nrt' ,  
        'ns' ,  
        'nt' ,  
        'nz' ,  
        'v' ,  
        'vd' ,  
        'vg' ,  
        'vi' ,  
        'vn' ,  
        'vq' ,  
    ]  
    content_raw = []  
    content=[]  
    # 读取语料库  
    with open(file_path, 'r',encoding="utf-8") as f:  
        content_raw=f.readlines()
```

```

f.close()

for i in content_raw:
    content.append( "".join(i.split("\n")[:-1]) )

#对文档进行分词处理
seg_list=[]
seg_flag_list=[]
for i in content:
    tt=psg.cut(i)
    ans1=[]
    ans2=[]
    for a1,a2 in tt:
        ans1.append(a1)
        ans2.append(a2)
    if len(ans1) != len(ans2):
        print("{}  {}".format(len(ans1),len(ans2)))
    seg_list.append( ans1 )
    seg_flag_list.append(ans2)
    # seg_list_many.append(seg_list)

#运用stopwords进行处理
corpus=[]

stopwords_raw=[]
stopwords=[]
with open("worddd.txt", encoding="utf-8") as f:
    stopwords_raw = f.readlines()
for i in stopwords_raw:
    stopwords.append( "".join(i.split("\n")[:-1]) )

for i in range(len(seg_list)):
    tmp=[]
    a=seg_list[i]
    b=seg_flag_list[i]
    if(len(a) != len(b)):
        print("False")
    for j in range(len(a)) :
        if a[j] not in stopwords and a[j] in tags:
            if b[j] in flags_we_wanted:
                try:
                    float(a[j])
                except:
                    tmp.append(a[j])
    tmp = ' '.join(tmp)
    corpus.append(tmp)

return corpus

```

导入包：

```
import pprint
from collections import Counter
import jieba
import numpy as np
import pandas as pd
import csv
import os
import jieba.posseg as pseg
import jieba.posseg as psg
import sys
import string
from sklearn import feature_extraction
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from gensim import similarities
import jieba.analyse
import gensim
from gensim import corpora
from gensim.models import LdaModel
from gensim.corpora import Dictionary
import pyLDAvis.gensim
import pyLDAvis.sklearn
import re
import math
from gensim.test.utils import common_corpus, common_dictionary
from gensim.models.ldamodel import LdaModel
from gensim.models.coherencemodel import CoherenceModel
import matplotlib.pyplot as plt
from pprint import pprint
import gensim
from sklearn.decomposition import LatentDirichletAllocation
import time
import random
```

操作部分：

```

train1 = []

# getTags()函数返回的是与transform_1()函数返回的字典一致的内容
data_lemmatized1=lemmatization("./alterContent_1.csv",getTags())

for line in data_lemmatized1:
    if line != '':
        line = line.split()
        train1.append([w for w in line])

id2word1 = corpora.Dictionary(train1)
#返回一个以列表形式存储的稀疏矩阵（词袋矩阵）
corpus1 = [id2word1.doc2bow(text) for text in train1]

```

### LDA模型主题数的确定：

为了确定LDA模型的主题数，我们查阅了网上的相关资料，然后在下面这篇博客找到了一个方法

#### 链接

利用博客中提到的困惑度perplexity，我们通过循环来尝试得到使得LDA模型具有最低困惑度的主题数

## 什么是Perplexity(困惑度)？

在信息论中，perplexity(困惑度)用来度量一个概率分布或概率模型预测样本的好坏程度。它也可以用来比较两个概率分布或概率模型。  
(译者：应该是比较两者在预测样本上的优劣) 低困惑度的概率分布模型或概率模型能更好地预测样本。

- 1.概率分布的困惑度
- 2.概率模型的困惑度

### 1.概率分布的困惑度

定义离散概率分布的困惑度如下：

$$2^{H(p)} = 2^{-\sum_x p(x) \log_2 p(x)}$$

其中 $H(p)$ 是概率分布 $p$ 的熵， $x$ 是样本点。因此一个随机变量 $X$ 的困惑度是定义在 $X$ 的概率分布上的（ $X$ 所有“可能”取值为 $x$ 的部分）。(译者： $x$ 不能包含零测集的点，不然 $p(x)$ 或 $\log_2 p(x)$ 没定义)

一个特殊的例子是 $k$ 面均匀骰子的概率分布，它的困惑度恰好是 $k$ 。一个拥有 $k$ 困惑度的随机变量有着和 $k$ 面均匀骰子一样多的不确定性，并且可以说该随机变量有着 $k$ 个困惑度的取值 ( $k$ -ways perplexed)。(在有限样本空间离散随机变量的概率分布中，均匀分布有着最大的熵)

困惑度有时也被用来衡量一个预测问题的难易程度。但这个方法不总是精确的。例如：在概率分布 $B(1, P=0.9)$ 中，即取得1的概率是0.9，取得0的概率是0.1。可以计算困惑度是：

$$2^{-0.9\log_2 0.9 - 0.1\log_2 0.1} = 1.38$$

同时自然地，我们预测下一样本点的策略将是：预测其取值为1，那么我们预测正确的概率是0.9。而困惑度的倒数是 $1/1.38=0.72$ 而不是0.9。(但当我们考虑 $k$ 面骰子上的均匀分布时，困惑度是 $k$ ，困惑度的倒数是 $1/k$ ，正好是预测正确的概率)

困惑度是信息熵的指数。

## 2. 概率模型的困惑度

用一个概率模型 $q$ 去估计真实概率分布 $p$ ，那么可以通过测试集中的样本来定义这个概率模型的困惑度。

$$b^{-\frac{1}{N} \sum_{i=1}^N \log_b q(x_i)}$$

其中测试样本 $x_1, x_2, \dots, x_N$ 是来自于真实概率分布 $p$ 的观测值， $b$ 通常取2。因此，低的困惑度表示 $q$ 对 $p$ 拟合的越好，当模型 $q$ 看到测试样本时，它会不会“感到”那么“困惑”。

我们指出，指数部分是交叉熵。

$$H(\hat{p}, q) = - \sum_x \hat{p}(x) \log_2 q(x)$$

其中  $\hat{p}$  表示我们对真实分布下样本点 $x$ 出现概率的估计。比如用 $p(x)=n/N$

最终我们打算利用如下公式计算困惑度**perplexity**：

$$\text{perplexity} = 2^{-\frac{\sum \log_2 p(w)}{N}}$$

其中，**p(w)**是指的测试集中出现的每一个词的概率，具体到LDA的模型中就是

$$p(w) = \sum z p(z | d) * p(w | z) p(w) = \sum z p(z | d) * p(w | z) p(w) = \sum z p(z | d) * p(w | z) \\ (z, d \text{ 分别指训练过的主题和测试集的各篇文档})$$

分母的N是测试集中出现的所有词，或者说是测试集的总长度，不排重。

计算**perplexity**的代码：

```
def perplexity(ldamodel, testset, dictionary, size_dictionary, num_topics):
    """calculate the perplexity of a lda-model"""
    # dictionary : {7822:'deferment', 1841:'circuitry', 19202:'fabianism'...}
    print ('the info of this ldamodel: \n')
    print ('num of testset: %s; size_dictionary: %s; num of topics: %s'%
    (len(testset), size_dictionary, num_topics))

    prep = 0.0
    prob_doc_sum = 0.0
    topic_word_list = [] # store the probability of topic-word: [(u'business',
    0.010020942661849608), (u'family', 0.0088027946271537413)...]

    for topic_id in range(num_topics):
        topic_word = ldamodel.show_topic(topic_id, size_dictionary)
        dic = {}
        for word, probability in topic_word:
            dic[word] = probability
        topic_word_list.append(dic)

    doc_topics_list = [] # store the doc-topic tuples: [(0, 0.0006211180124223594), (1,
    0.0006211180124223594), ...]

    for doc in testset:
```

```

    doc_topics_list.append(ldamodel.get_document_topics(doc,
minimum_probability=0))

    testset_word_num = 0
    for i in range(len(testset)):
        prob_doc = 0.0 # the probability of the doc
        doc = testset[i]
        doc_word_num = 0 # the num of words in the doc
        for word_id, num in doc:
            prob_word = 0.0 # the probability of the word
            doc_word_num += num
            word = dictionary[word_id]
            for topic_id in range(num_topics):
                # cal p(w) : p(w) = sumz(p(z)*p(w|z))
                prob_topic = doc_topics_list[i][topic_id][1]
                prob_topic_word = topic_word_list[topic_id][word]
                prob_word += prob_topic*prob_topic_word
            prob_doc += math.log2(prob_word) # p(d) = sum(log(p(w)))
        prob_doc_sum += prob_doc
        testset_word_num += doc_word_num
    prep = 2**(-prob_doc_sum/testset_word_num) # perplexity = exp(-
sum(p(d)/sum(Nd)))
    print ("the perplexity of this ldamodel is : %s"%prep)
    return prep

```

我们以1300条数据作为训练集，以剩下的数据作为测试集，在主题数 $k = 1 \rightarrow 20$ 中寻找最佳主题数

运行如下代码：

```

x=list(range(0,len(corpus),1))
random.shuffle(x)
x1=x[0:1300]
x2=x[1300:-1]
d1=[]#训练集
d2=[]#测试集
for i in x1:
    d1.append(corpus[i])
for i in x2:
    d2.append(corpus[i])
ans=[]
num=list(range(1,20,1))
testset=[]
for i in range(int(len(corpus)/3)):
    testset.append(corpus[i*3])
for i in range(1,20,1):
    lda_model = gensim.models.ldamodel.LdaModel(corpus=d1,

```

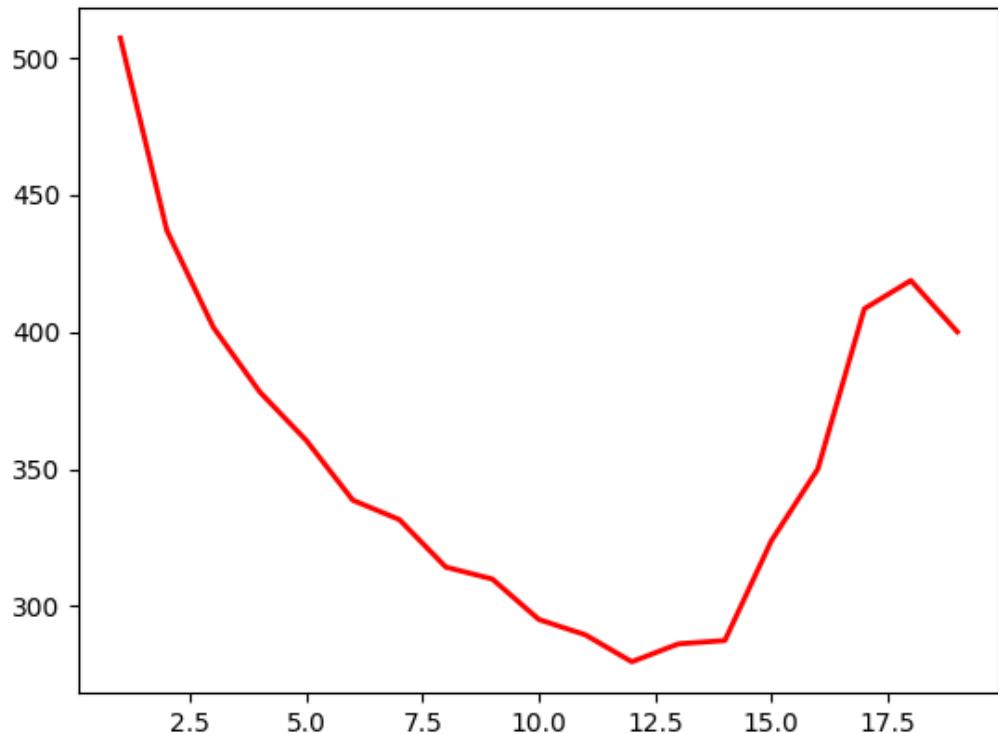
```

        id2word=id2word,
        num_topics=i,
        random_state=100,
        update_every=1,
        chunksize=100,
        passes=20,
        alpha='auto',
        per_word_topics=True)

# a=2**(-lda_model.log_perplexity(d2))
a=perplexity(lda_model,d2,id2word,len(id2word),i)
b=lda_model.get_topics()
print("\n{}  Perplexity: {}".format(i,a))
ans.append(a)
plt.plot(num,ans, 'r-', linewidth = 2 , markersize = 4)
plt.show()

```

得到如下图像



(图像最低点好象是12,之前做的时候看走眼成11了😊)

### 以11作为主题数的LDA模型

然后我们以11作为主题数, 得到了一个LDA模型

```

lda_model = gensim.models.ldamodel.LdaModel(corpus=corpus1,

```

```

    id2word=id2word1,
    num_topics=11,
    #以下参数可以参考gensim的帮助文档, 这里就是直接用

    的网上的

    #有可能会有些。。。问题
    #做的时候没遇到问题, 调参太麻烦了

    random_state=100,
    update_every=1,
    chunksize=100,
    passes=20,
    alpha='auto',
    per_word_topics=True)

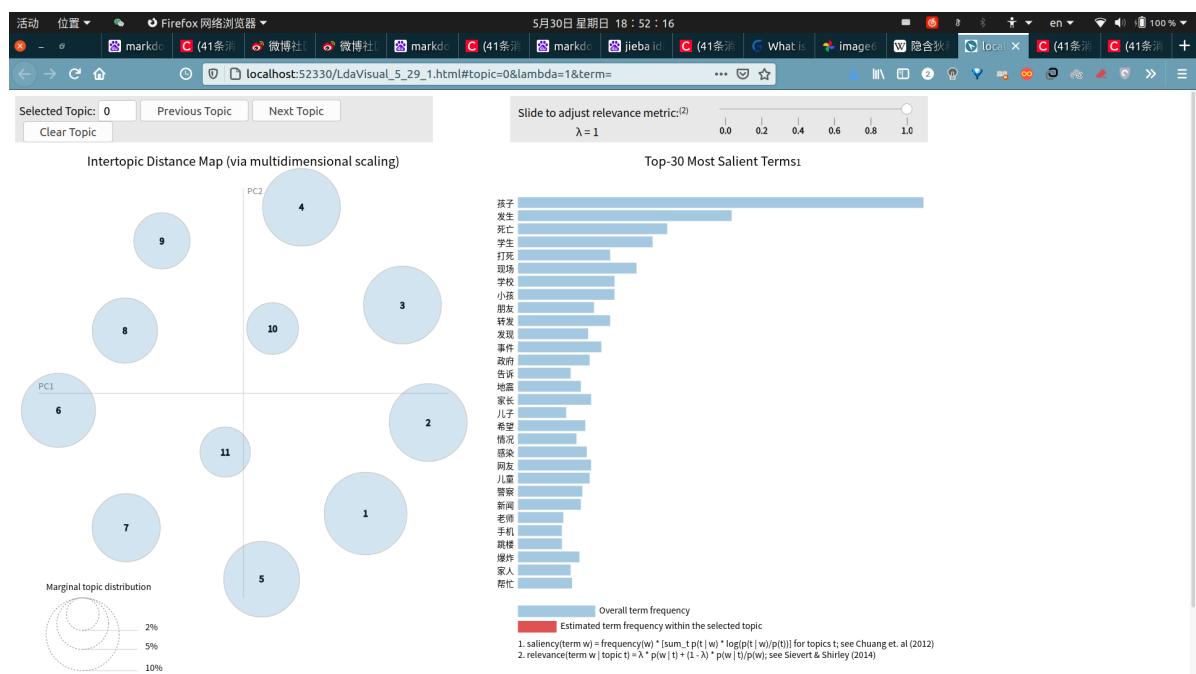
#生成LDA模型的可视化html报告

vis = pyLDAvis.gensim.prepare(lda_model, corpus1, id2word1, mds='mmds')
pyLDAvis.save_html(vis, './LdaVisual_5_29_1.html')

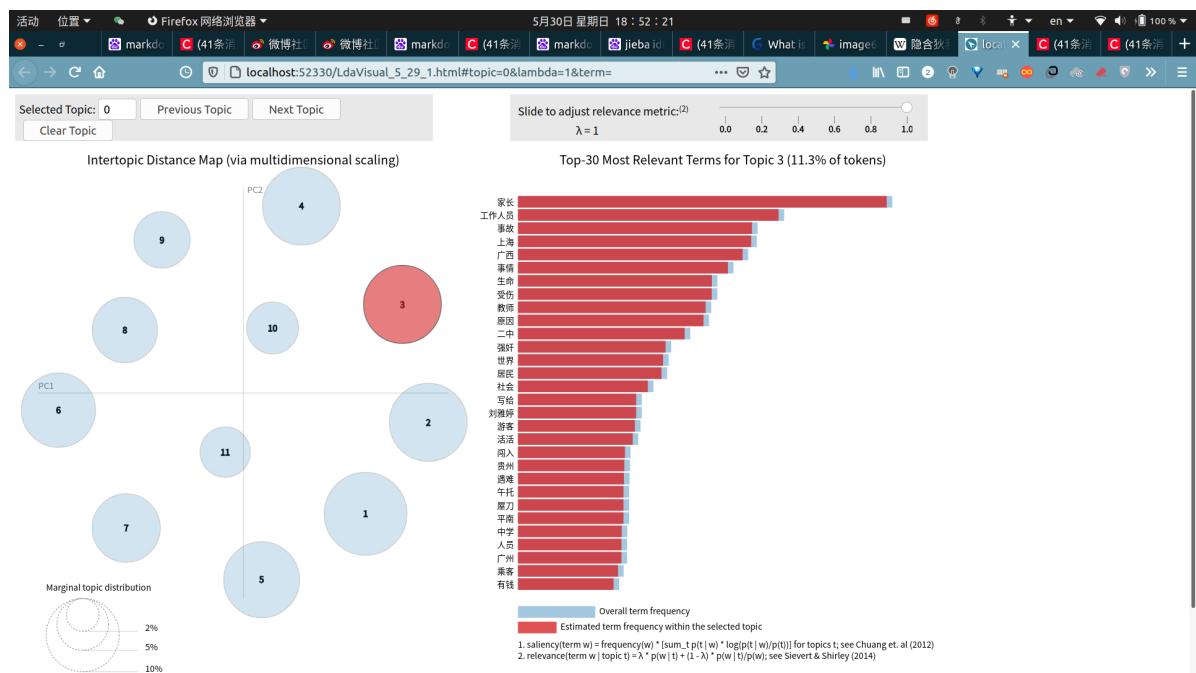
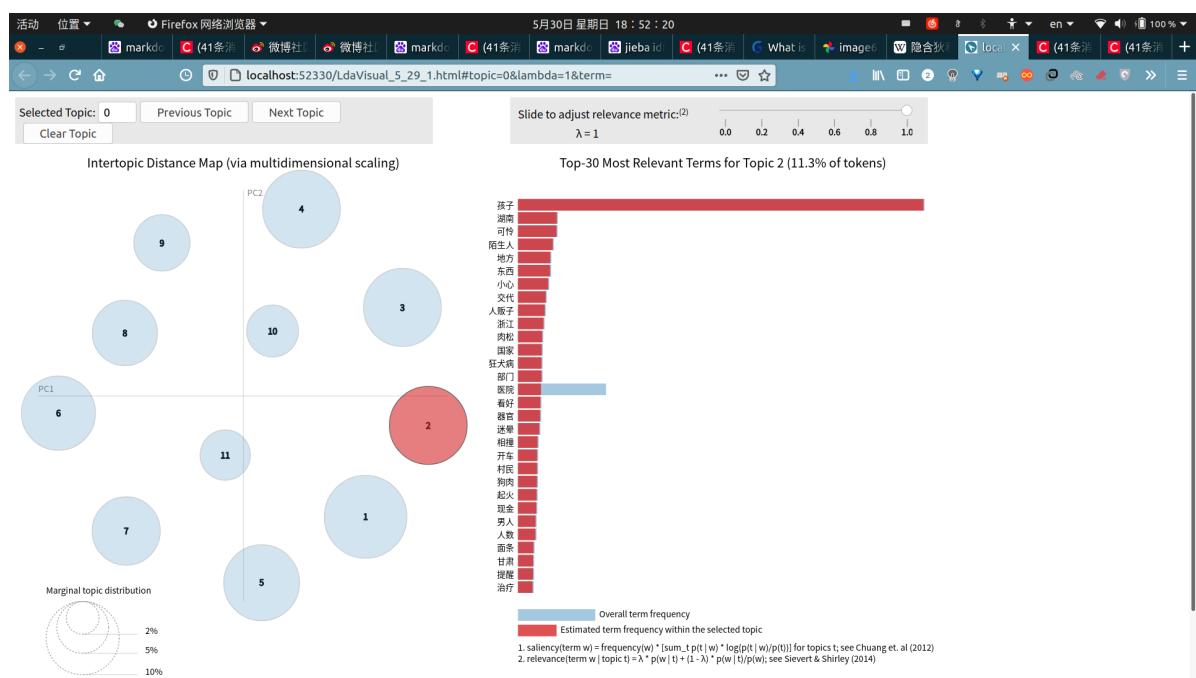
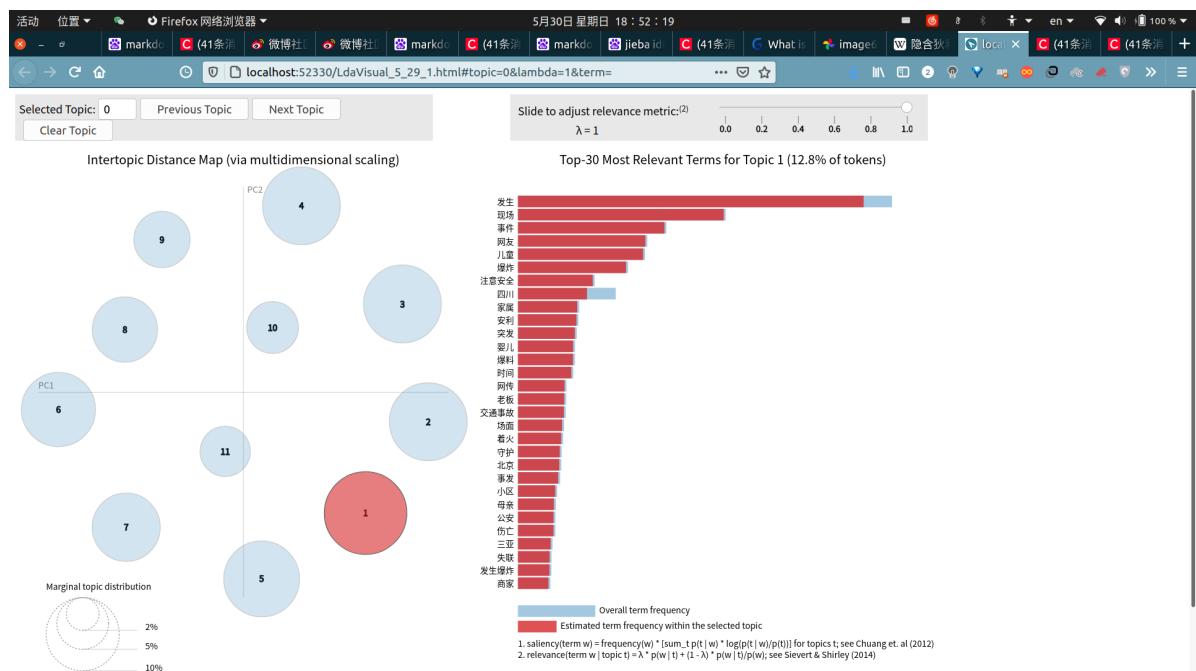
```

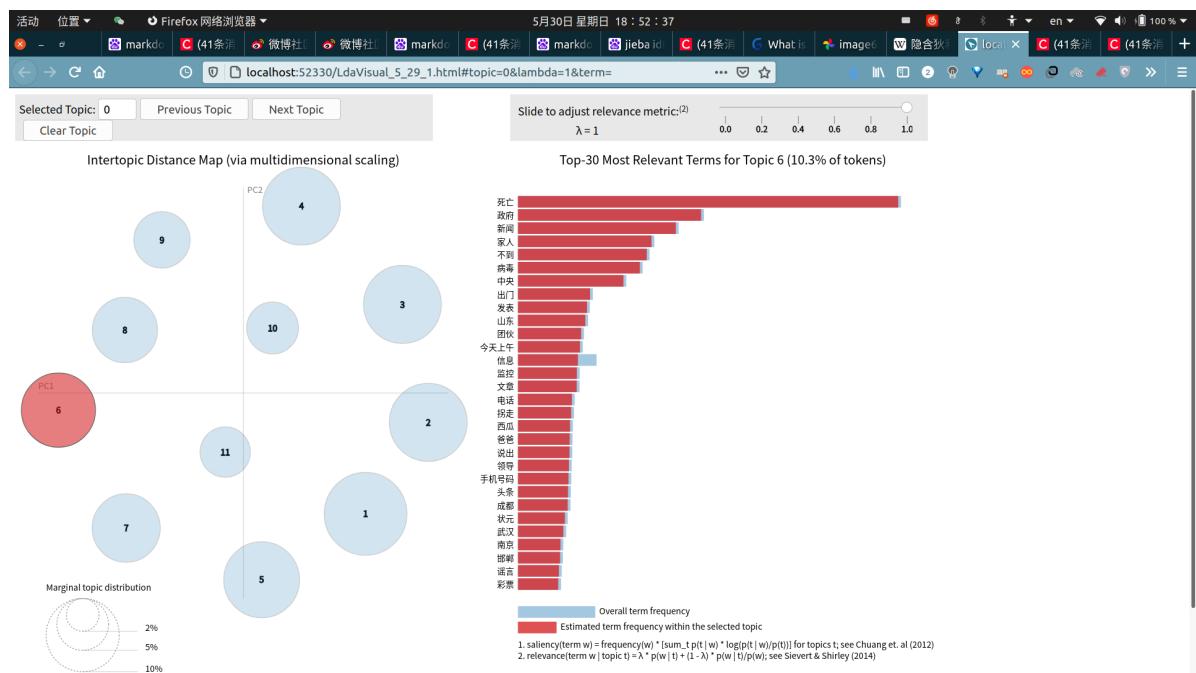
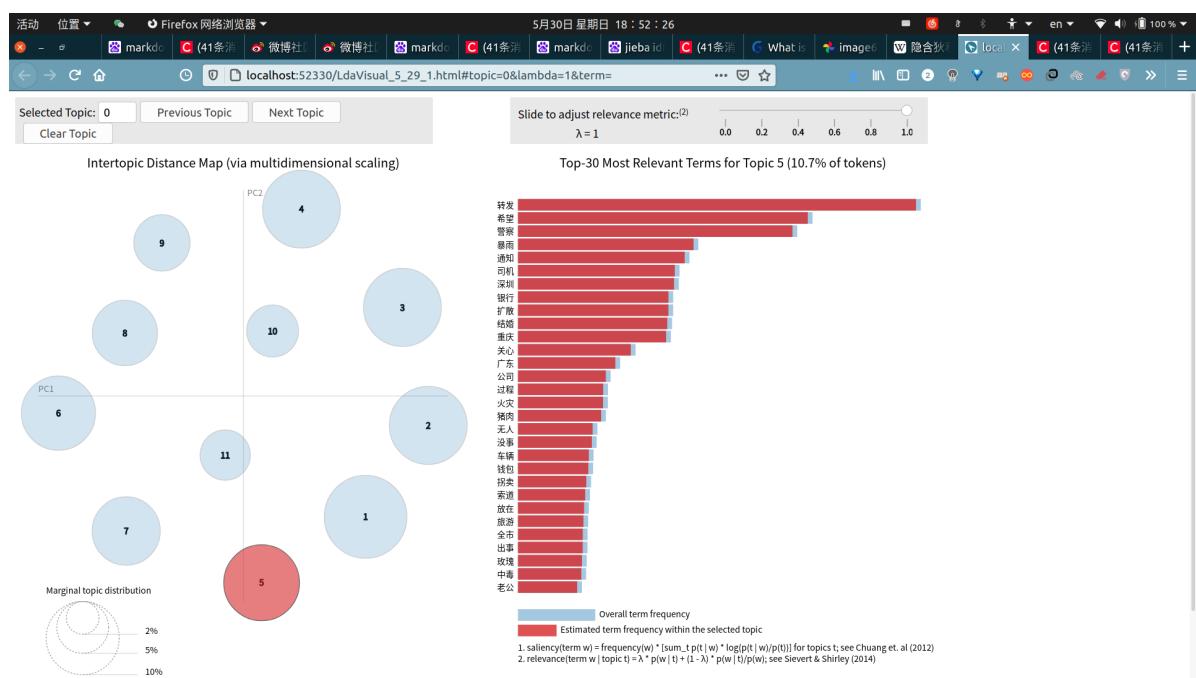
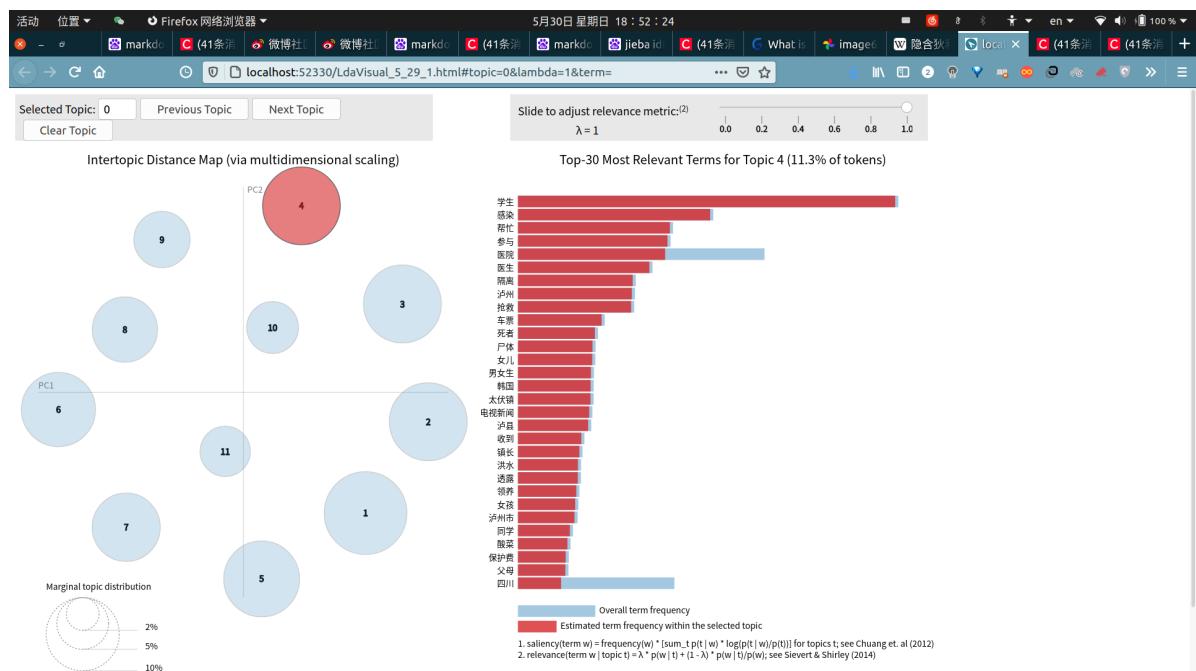
## LDA模型的可视化

- 前三十个高频词



- 下面是主题1-6的前三十个最相关词语的展示图

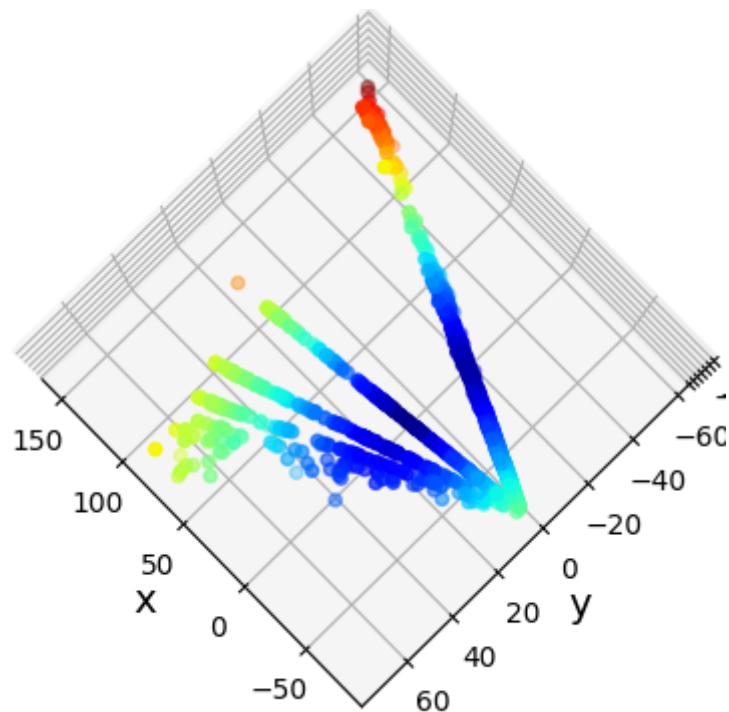
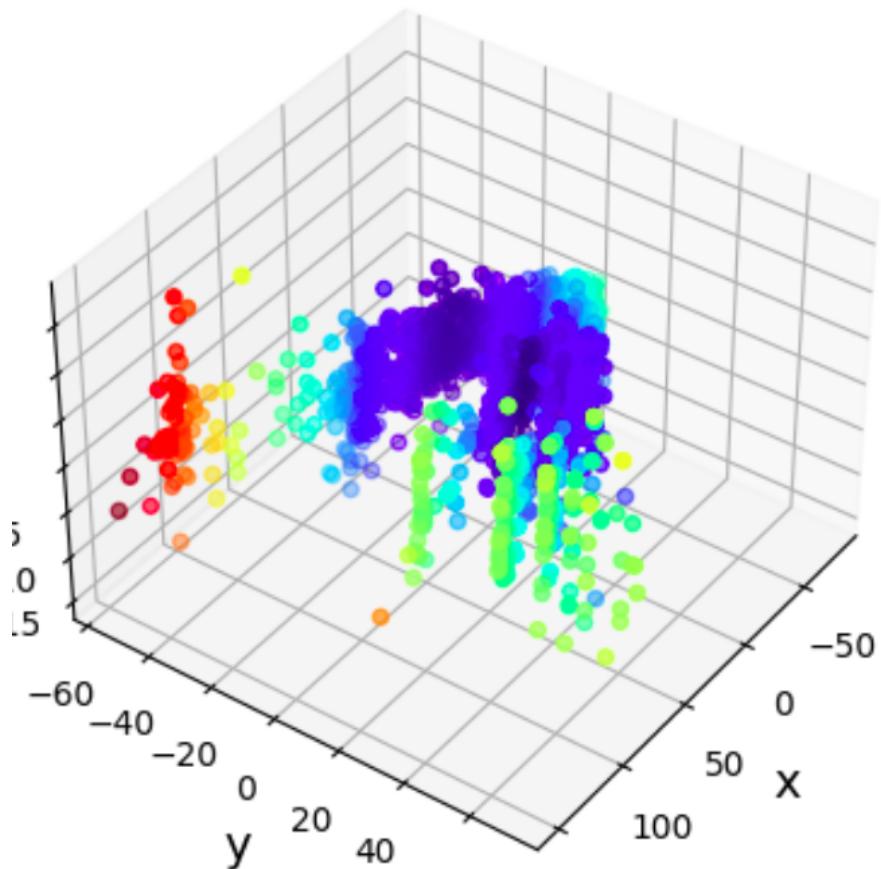




## ● 基于数据特征的GMM分类

### 1. 对数据特征利用PCA降维

首先，我们利用PCA将数据特征降到3维（颜色是一个与坐标有关的函数）：

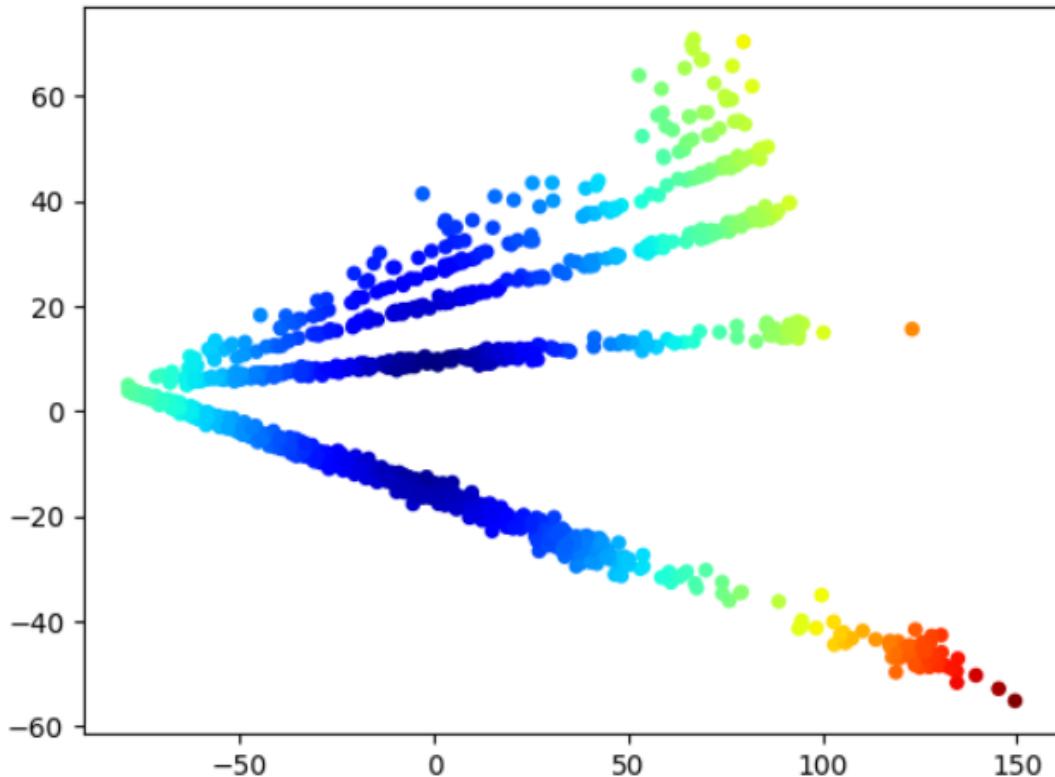


可以看到基本上所有的点都被划分到了一个个平面上

```
chenzhean@czacomputer:~/Desktop/Files/main_focus/python_大数据技术$  
s/python_大数据技术 ; /usr/bin/env /usr/bin/python3.8 /home/chenzhe  
5.842923320/pythonFiles/lib/python/debugpy/launcher 36439 -- /home/  
数据技术/5_10_爬虫/train_5_16.py  
pca.explained_variance_ratio_:[0.83956486 0.15629385 0.00358973]  
pca.explained_variance:[2567.95393136 478.05168439 10.97982164]
```

输出的方差膨胀因子也让我们更有底气将数据特征降到2维

因此，我们将数据的特征通过两个主成分来代替。画出我们每条微博数据在主成分构成的空间中的散点图，如下图所示：



图中的各个点之间有明显的聚类趋势，其中有四组点呈现明显的聚集情况，这些同类的点在二维空间分布上有明显的线性趋势；另外有部分点较为分散，但仍然有聚集的趋势。

## 2.k-means聚类（舍弃）

根据图像中各组点的分布规律，我们进行聚类分析。首先使用K-means聚类（类别划分为5类），分析发现，聚类效果并不符合直观上的各组点的分布情况（K-means聚类结果如下图）。

### K-means classification

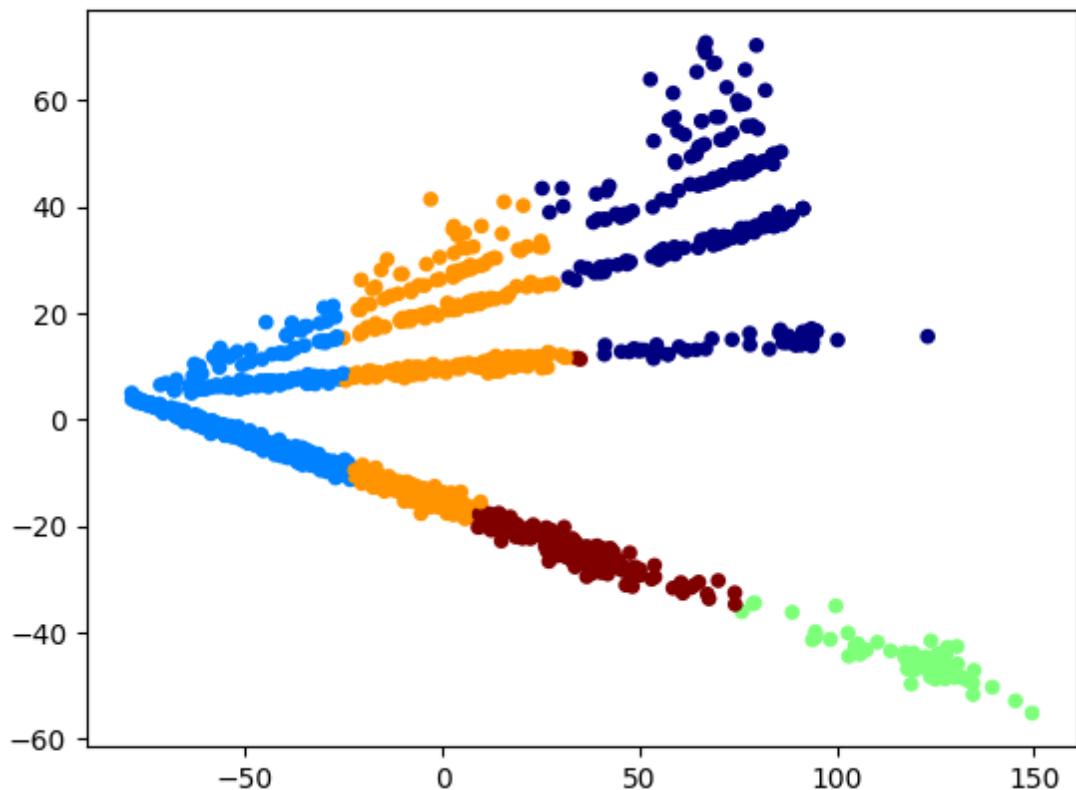


图3.K-means分类、同一类用相同颜色表示，图中分类与实际分类趋势明显不相符

```
def k_means(weight, num):  
    ...  
    注意！！！  
    这里的weight不能直接传词袋矩阵，要传类似特征矩阵的东西  
    ...  
    from sklearn.cluster import KMeans  
  
    clusters = KMeans(  
        n_clusters=num,  
        init='k-means++'  
    ) # 设置聚类模型  
  
    # 每个样本所属的簇  
    label = clusters.fit_predict(weight) # 把weight矩阵扔进去fit一下,输出label  
  
    # 用来评估簇的个数是否合适,距离约小说明簇分得越好,选取临界点的簇的个数  
    print(clusters.inertia_)  
  
    return label, clusters.inertia_
```

### 3.高斯混合分布GMM模型

经过我们的调研后发现，K-means聚类在二维空间上，是通过确定圆点和相应的半径，用圆形去覆盖空间中的样本点，不适合于我们当前数据的这种离散的线性分布，因此我们考虑使用高斯混合模型来进行分类。所谓高斯混合模型，本质就是融合几个单高斯模型，来使得模型更加复杂，从而产生更复杂的样本。理论上，如果某个混合高斯模型融合的高斯模型个数足够多，它们之间的权重设定得足够合理，这个混合模型可以拟合任意分布的样本。它的具体形式为：

假设混合高斯模型由K个高斯模型组成（即数据包含K个类），则GMM的概率密度函数如下：

其中，是第k个高斯模型的概率密度函数，可以看成选定第k个模型后，该模型产生x的概率；是第k个高斯模型的权重，称作选择第k个模型的先验概率，且满足。

下面是具体实现GMM以及画图展示的代码

```
#。。。。省去了一些用过很多次的包的导入，需要的话上面自取
from sklearn.datasets import make_blobs
from sklearn.mixture import GaussianMixture as GMM

#导入特征数据，pca降维
data_1=pd.read_csv("features.csv").to_numpy()
X,e,f=func_pca(data_1,2)

from matplotlib.patches import Ellipse
def draw_ellipse(position, covariance, ax=None, **kwargs):
    """用给定的位置和协方差画一个椭圆"""
    ax = ax or plt.gca()

    #将协方差转换为主轴
    if covariance.shape == (2, 2):
        U, s, Vt = np.linalg.svd(covariance)
        angle = np.degrees(np.arctan2(U[1, 0], U[0, 0]))
        width, height = 2 * np.sqrt(s)
    else:
        angle = 0
        width, height = 2 * np.sqrt(covariance)

    #画出椭圆
    for nsig in range(1, 4):
        ax.add_patch(Ellipse(position, nsig * width, nsig * height,
                             angle, color=[0.02,0.02,0.02],**kwargs))

def plot_gmm(gmm, X, label=True, ax=None):
    ...
    对于gmm的结果进行画图展示
```

```

    ...
    ax = ax or plt.gca()
    labels = gmm.fit(X).predict(X)
    if label:
        ax.scatter(X[:, 0], X[:, 1], c=labels, s=5, cmap='viridis', zorder=2)
    else:
        ax.scatter(X[:, 0], X[:, 1], s=5, zorder=2)
    ax.axis('equal')

    w_factor = 0.2 / gmm.weights_.max()
    for pos, covar, w in zip(gmm.means_, gmm.covariances_, gmm.weights_):
        draw_ellipse(pos, covar, alpha=w * w_factor)

#用椭圆形来拟合数据
gmm = GMM(n_components=5, init_params="random", max_iter=1000).fit(X)
plot_gmm(gmm, X)

```

我们最终的聚类效果如下图所示：

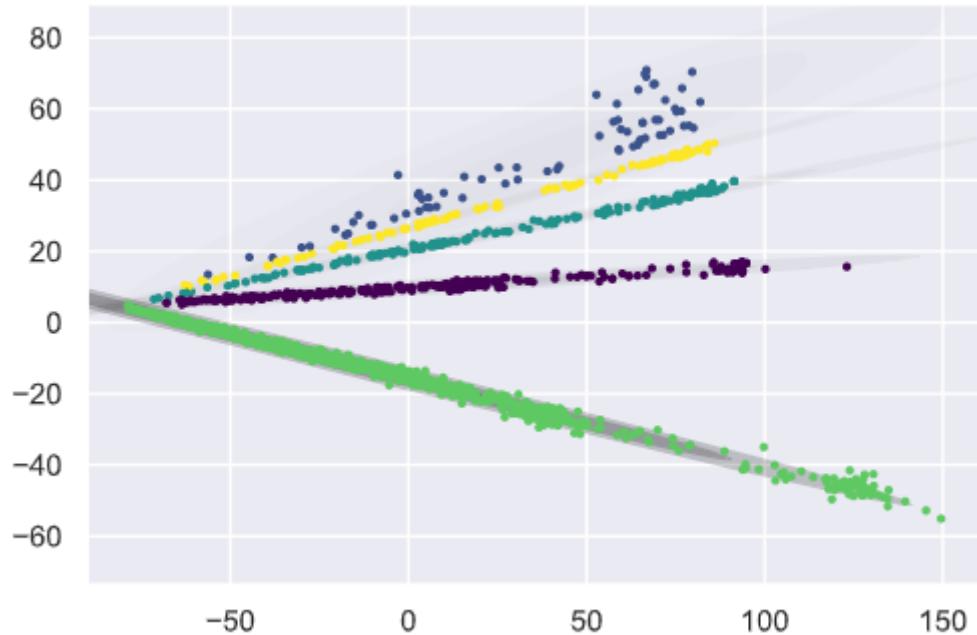


图 4. *GMM* 分类结果，同一类用相同颜色表示

就算所有参数都已经确定的情况下，这里得出的GMM的分类也不是确定的，因为一开始的时候初始条件不同会导致迭代后产生不同的分类效果，上面的这张图是我们不断运行下面这段代码，偶然间得到的

```

#用椭圆形来拟合数据
gmm = GMM(n_components=5, init_params="random", max_iter=1000).fit(X)
plot_gmm(gmm, X)

```

从图4中看出，模型很好的聚集成5类数据，且每组数据聚类趋势明显。我们用标号0~4表示5个类的类别名称，输出每条微博正文最终分析的类别结果(实际中输出到了**labels.txt**中，下面只是效果图)。

下图为部分微博正文与主题类别的关系：

L	M	类型
不实信息具体内容		3
不爱请别伤害，更何况这是在工作的工作犬 #中国正能量#		3
这是真的吗？		3
潍坊风筝节，大人没有看好孩子，风筝尾巴没有成人管理，小孩脖子缠住上拉上天了，颈椎断了，活不了了。		3
网传陕西西安一哥在车内猝死。随后，交警闻讯后及时赶到事发现场，给出租车贴上了违停罚单……视频里有语音，自行判断。		0
五一的泰山，这是去人体按摩的吧		3
#印度警民街头混战# 印度是个很奇葩的国度，人口数量仅次于我国，国土面积排世界第七……印度的火车竟然能买“挂票”？[费解]		3
#青岛1人阳性：由南京坐高铁返青#冷链样本51份检出阳性！青岛再现感染者，盘点疫情后续#今日热门#		3
#太原理工大学老师# ❶❷上课放错视频 哈哈哈哈囧 这老师品味还不错[污][污][污]		3
4月7日，网传长沙一民警街头过肩摔女子…？来看看，这才是真相！ #视频制作人出道计划# [思考]#剧情制作人计划#		0
4月1日，是什么节日？今天中国第一个消防战士纪念日。		0
#江苏大学开除遣送多名留学生#江苏徐州：一高校国际学院多名留学生吸毒被遣返回国#今日热门#		3
因为送避孕套迟到八分钟导致意外怀孕，砀山女子把外卖小哥告上法庭，要求赔偿各项损失共计三万余元……难人 #走心视频#		3

图5.部分分类结果

我们对每类数据进行时间序列分析。首先画出数据时序图，以时间截为横坐标（时间截从201604到202005），做出每月该类微博谣言数量随时间变化的折线图。如图6所示。可以看到谣言数量随时间做周期变化，对原数据做ADF检验，发现检验结果p值为0.065，说明原数据不是纯随机性数据，因此对数据进行差分处理。

以第三类数据为例，我们可以看到第三类数据主要和“孩子”，“死亡”等关键词有关，主题是青少年和死亡。第三类数据在时序图中由红色曲线表示出。做出的一阶差分结果和二阶差分结果如图9。

再对差分后的数据做ADF检验，发现一阶差分后p值已经小于0.05，并且从图中看出，一阶差分与二阶差分相比随机性变动不大，因此我们使用一阶差分的数据，并采用ARIMA模型对数据进行分析。

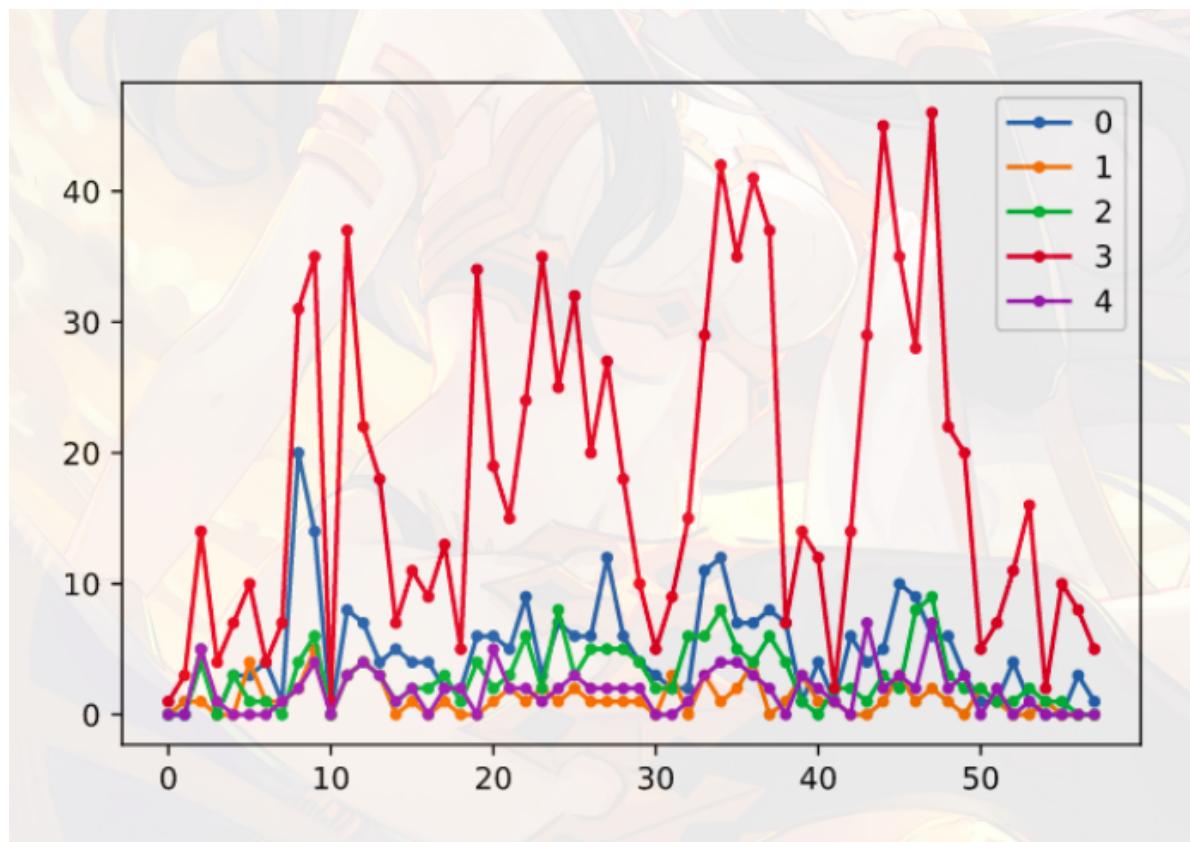


图6.每类谣言数量随时间变化趋势



图7.第三类谣言的关键词

```
> adf.test(rate)
Augmented Dickey-Fuller Test

data: rate
Dickey-Fuller = -3.4258, Lag order = 3, p-value = 0.06059
alternative hypothesis: stationary

> adf.test(diff(rate))
Augmented Dickey-Fuller Test

data: diff(rate)
Dickey-Fuller = -3.645, Lag order = 3, p-value = 0.03732
alternative hypothesis: stationary

> adf.test(diff(diff(rate)))
Augmented Dickey-Fuller Test

data: diff(diff(rate))
Dickey-Fuller = -5.1539, Lag order = 3, p-value = 0.01
alternative hypothesis: stationary
```

图8.原始数据、一阶差分和二阶差分的 $ADF$ 检验结果

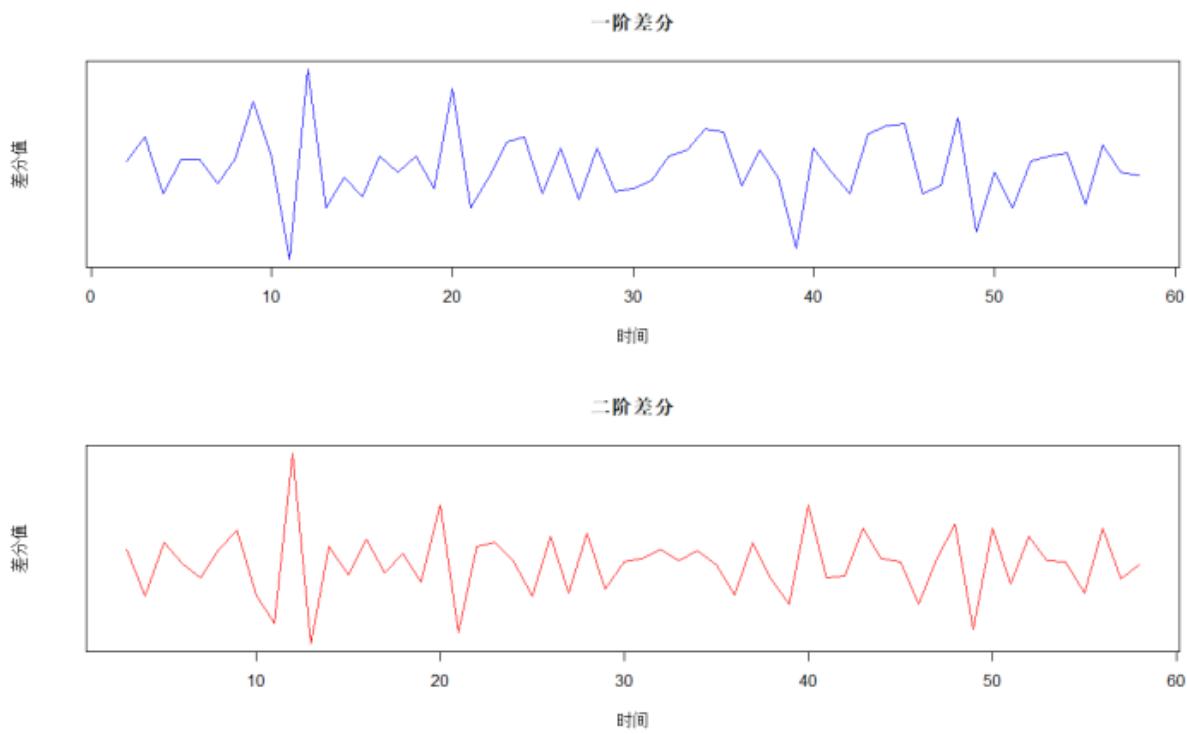


图9.一阶差分和二阶差分结果图

对于一阶差分后的数据，我们判别其ACF与PACF的分布情况。从图10中可以看出，ACF主要在取1时超出标准范围，PACF主要在取值为1、2时超出标准范围，因此我们讨论一下几个模型：ARIMA(1,1,1)，ARIMA(1,1,2)和ARIMA(1,1,3)

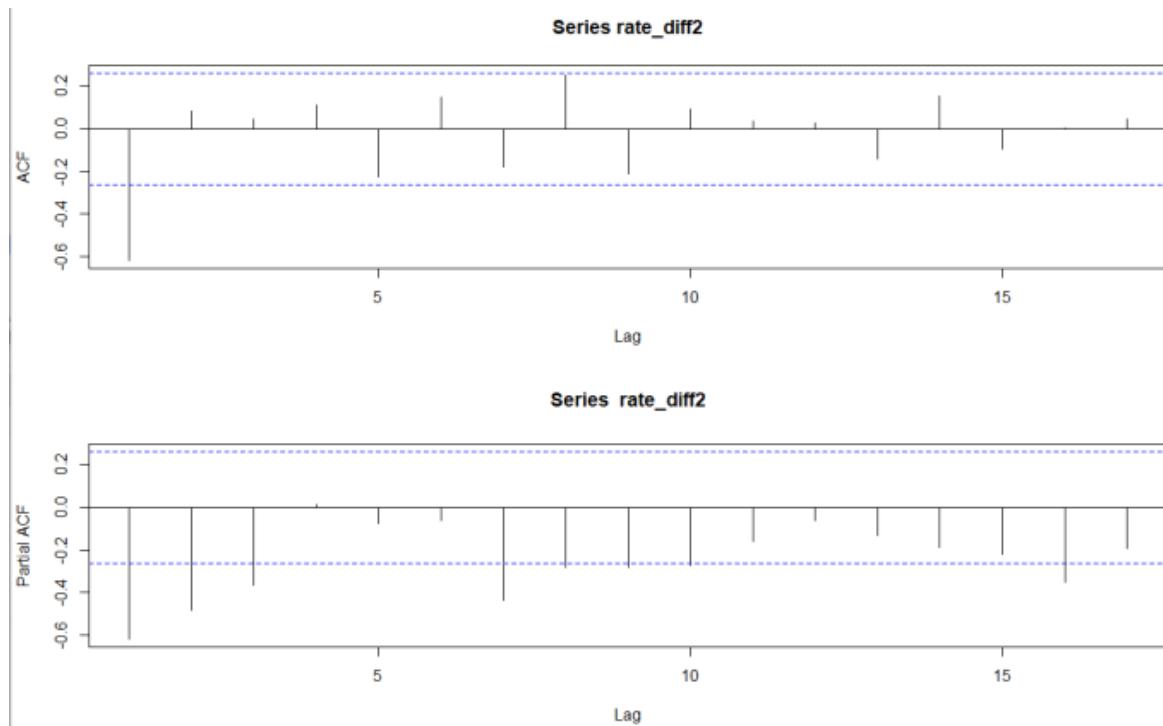


图10.AC F与PAC F分析

我们采用AIC与BIC得分来判定模型效果的好坏，得分越低，效果越好，测评如下。

```

> ##ARIMA(1,1,1)
> rate.fit1<-arima(rate,order=c(1,1,1))
> AIC(rate.fit1)
[1] 446.0468
> BIC(rate.fit1)
[1] 452.1759
>
> ##ARIMA(1,1,2)
> rate.fit2<-arima(rate,order=c(1,1,2))
> AIC(rate.fit2)
[1] 447.3532
> BIC(rate.fit2)
[1] 455.5254
>
> ## ARIMA(1,1,3)
> rate.fit3<-arima(rate,order=c(1,1,3))
> AIC(rate.fit3)
[1] 448.8711
> BIC(rate.fit3)
[1] 459.0864

```

图 11. 模型测试结果

可以看出，模型 $ARIMA(1,1,1)$ 效果最好。进一步对模型检验，检验模型的残差是否时白噪声。从图 12 和图 13 可以看出，通过 qq 图展示结果说明残差服从正态分布，而 Ljung-Box 检验证明残差为白噪声。

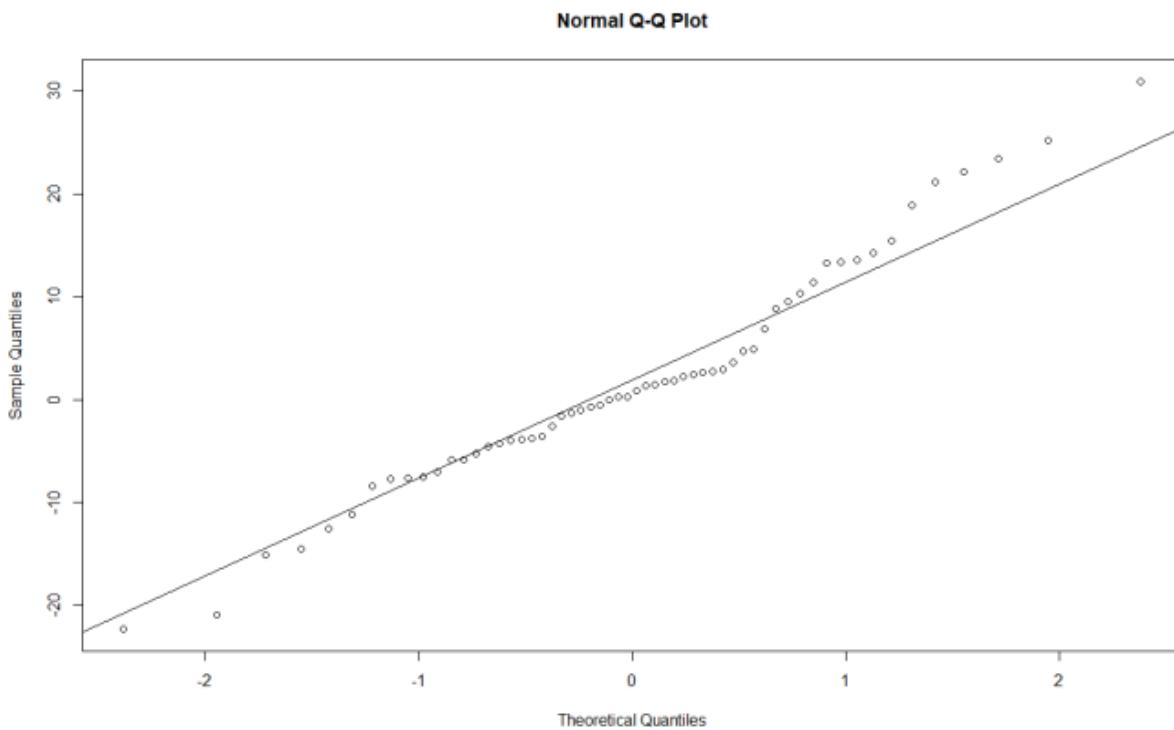


图 12. 模型残差的 qq 图

```

Box-Ljung test

data: rate.fit1$residual
X-squared = 0.84381, df = 2, p-value = 0.6558

Box-Ljung test

data: rate.fit1$residual
X-squared = 1.415, df = 3, p-value = 0.702

Box-Ljung test

data: rate.fit1$residual
X-squared = 1.4238, df = 4, p-value = 0.84

```

图 13. *Box-Ljung* 检验结果

利用ARIMA模型进行预测，可以看到未来几个月可能出现的该类谣言数量趋势。

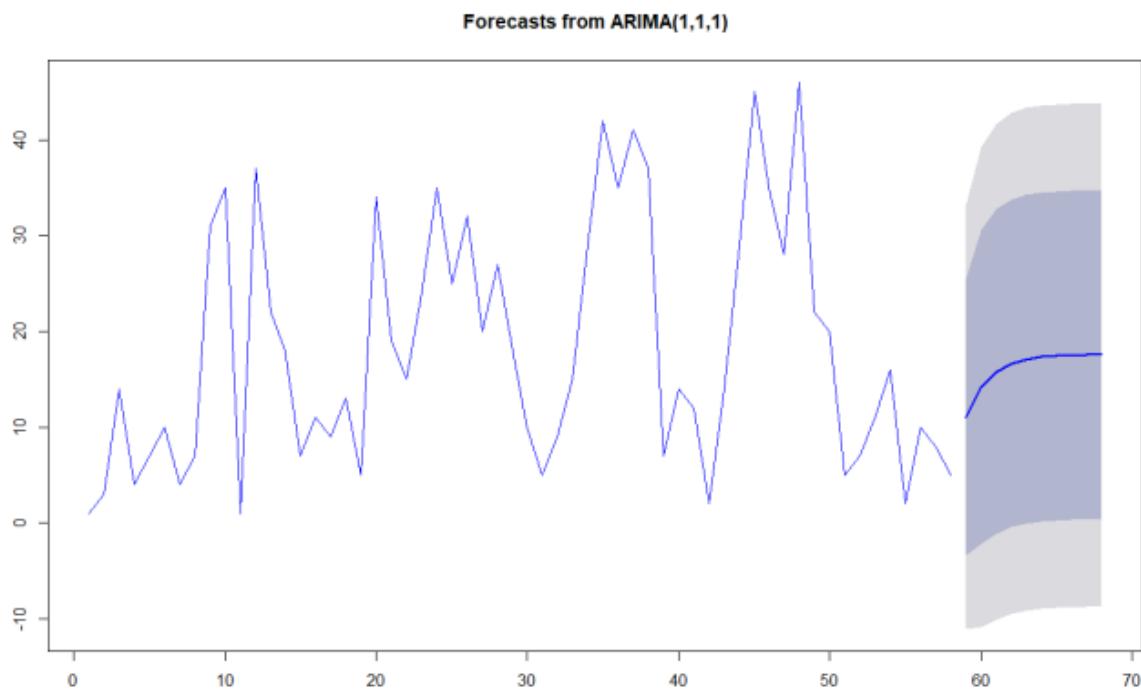


图 14. *ARIMA* 模型的预测结果

## 选做部分

### 正常微博爬取过程

有了上面关于爬取等量正常信息的经验，稍微修改代码过后，就可以得到爬取正常微博的代码。

在爬取正常信息的时候，我们并没有直接爬取之前不实信息发布者的正常微博，具体操作如下：

- 假设原来没有经过去重的爬取数据以用户信息来划分后的集合（无重复）为  $a$ ，而经过去重之后剩下的不实微博数据为以用户信息来划分后的集合（无重复）  $b$ ，然后我们获得差集  $c = a - b$

- 在差集  $c$  所包含的用户中，看情况每个人爬取一条正常的微博信息

由于一部分特征我们之前第一次爬虫已经获取到了

```
["被举报人", "性别", "地区", "信用等级", "是否会员"]
```

所以我们准备爬取的是如下内容：

```
["数据", "数据精确", "isHyperlink"]
```

我们这么处理的原因如下：对于我们确定的那9个数据特征，如果再去获取去重之后剩下的那些用户的正常信息，那么在性别，地区等特征上就会导致重复。

与正常微博信息信息爬取有关的代码如下：

Note:

- `all_5_10.csv`是爬取的所有不实信息
- `5_10_corrected_3.csv`是经过多次去重之后得到的数据

```
all_data=pd.read_csv("all_5_10.csv")
data=pd.read_csv("5_10_corrected_3.csv")
all_data1=all_data["被举报人的微博主页"]
all_data2=all_data["被举报内容的微博网址"]
all_data3=all_data["被举报人"]
data1=data["被举报人的微博主页"]
data2=data["被举报内容的微博网址"]
data3=data["被举报人"]
cross_data3=list(set(all_data3)^set(data3))
cross_data1=list(set(all_data1)^set(data1))
```

下面是爬取正常微博的函数`right_get()`的代码

```
def right_get(driver,href,num):
    try:
        driver.get(href)
    except:
        driver.execute_script('window.stop ? window.stop() :
document.execCommand("Stop");')
    count=1
    while(count):
        if count > 2:
```

```

        break

    try:
        locator=(By.XPATH,"//a[@node-type='feed_list_item_date']")
        element =
    WebDriverWait(driver,3).until(EC.presence_of_element_located(locator))
        break
    except:
        driver.refresh()
        count+=1
        continue

# tt获取所有的“框”
tt=driver.find_elements_by_xpath("//a[@node-type='feed_list_item_date']")
all_href=[]
for i in tt:
    all_href.append(i.get_attribute("href"))
# print(len(tt))
count=0
message=[]
for i in all_href:
    try:
        driver.get(i)
    except:
        driver.execute_script('window.stop ? window.stop() :
document.execCommand("Stop");')
    if count>=1:
        break
    locator=(By.XPATH,"//div[@class='WB_detail']//div[@class='WB_text W_f14']")
    while(1):
        try:
            element =
        WebDriverWait(driver,3).until(EC.presence_of_element_located(locator))
            break
        except:
            driver.refresh()
            continue
        try:
            driver.find_element_by_xpath("//div[@class='W_tips tips_rederror
clearfix']")
            continue
        except:
            is_hyperlink=0
            #a是微博正文中带有的<a>标签的东西

a=driver.find_elements_by_xpath("//div[@class='WB_detail']//div[@class='WB_text
W_f14']//a")
#b是微博正文

b=driver.find_element_by_xpath("//div[@class='WB_detail']//div[@class='WB_text
W_f14']").text

```

```

        for i in a:
            if "L" in i.text:
                is_hyperlink=1
                b_raw=b
        for i in a:
            b="".join(b.split(i.text))
        if not b_raw:
            message.append("###")
            message.append("###")
            message.append("###")
        else:
            message.append(b)
            message.append(b_raw)
            message.append(is_hyperlink)
        count+=1
        print("第{}条数据 第{}条正常原文".format(num,count))
    return message

```

上面的函数外层还有嵌套一层函数，也就是我们的主函数

```

def func(all_data,data1,start,end,index,cookies):
    driver=start_webdriver()
    driver.get('https://weibo.com/login.php')
    for i in cookies:
        driver.add_cookie(cookie_dict=i)
    driver.set_page_load_timeout(60)
    all_message=[]

    count_txt="count_"+str(index)+".txt"
    with open(count_txt, "a+", newline="") as g:
        writer1 = csv.writer(g)
        writer1.writerow(["第{}号进程正式开始于: {}".format(index,time.ctime())])
        print("第{}号进程正式开始于: {}".format(index,time.ctime()))
        g.close()

    with open("new_information_"+str(index)+".csv",'a+')as f:
        writer=csv.writer(f)
        writer.writerow(["被举报人","性别","地区","信用等级","是否会员","数据","数据精确","isHyperlink"])
        f.close()

    for i in range(start,end):
        # switch_to_active(driver)
        # time.sleep(1)

        while(1):
            try:
                a=right_get(driver,data1[i],i)
                break
            except:

```

```

        continue

    if a:
        kkk=list(all_data[all_data["被举报人的微博主页"]==data1[i]].iloc[0])
        all_message.append(kkk[3:6]+kkk[8:10]+a)
        # break

    else:
        all_message.append([])

    print("@@@@@第{}条数据已经成功".format(i))
    with open(count_txt, "a+", newline="") as g:
        writer1 = csv.writer(g)
        writer1.writerow(["第{}条已爬完".format(i)])
        g.close()

    #在这里我们选择每爬取十条正常微博信息，或者到达end了，就写入一次
    if ((i+1) % 10 ==0 or (i+1) == end):
        with open("new_information_"+str(index)+".csv",'a+')as f:
            writer=csv.writer(f)
            for t in all_message:
                # for r in range(len(t)):
                #     if t[r]=="":
                #         t[r]=="##"
                # for k in range(len(t),3):
                #     t.append("##")
                writer.writerow(t)
                # tmp_all_message.append(t)
            f.close()
        all_message=[]

```

由于时间原因，最后我们开了两个进程，获取到了736条正常信息微博数据，命名为“new\_info.csv”

## 🌐 特征获取

由于前文已经讲过如何获取不实信息的特征，这里采用的是一样的方法，就不再过多叙述。

我们处理过后的正常信息微博的特征文件为：“alter\_right\_message\_features.csv”

## 🌐 支持向量机

### 支持向量机的原理简述

支持向量机（SVM），实质上就是在一个n维空间中存在的一组点，在这组点之间生成一个n-1维的超平面，将这组点分割为两部分，每部分点到这一超平面的距离之和达到最小，而求解这一超平面的过程就是建立SVM模型的过程，模型的最终成果就是这一超平面。

具体到我们的实验中，我们首先将新的数据集随机排序后，按照训练集：验证集：测试集=8：1：1的比例进行划分，用于支持向量机的训练和验证。

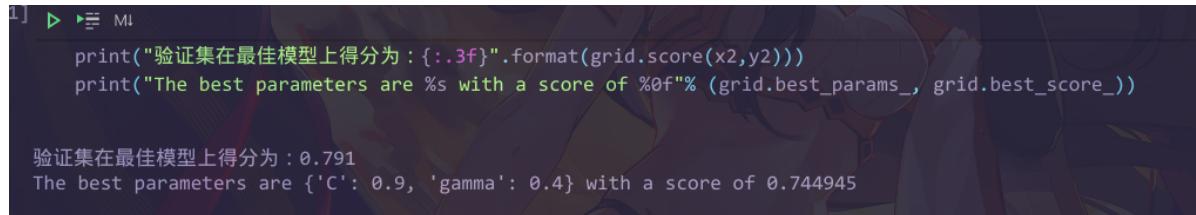
在我们的训练集上，通过对训练集不同内容的提取，训练出100个模型。

这些模型的参数，分别是：

- 松弛变量系数C（惩罚系数）从0.1到1，每隔0.1取一个值
- gamma参数从0.1到1

两两组合共组成100个模型

利用我们的测试集，我们检验一下模型的拟合效果，发现松弛变量取0.9, gamma取值0.4时模型拟合效果最好，正确率可以达到79%。下图为模型最佳参数的测试结果：



```
print("验证集在最佳模型上得分为 :{:.3f}".format(grid.score(x2,y2)))
print("The best parameters are %s with a score of %0f%% (grid.best_params_, grid.best_score_))
```

验证集在最佳模型上得分为 :0.791  
The best parameters are {'C': 0.9, 'gamma': 0.4} with a score of 0.744945

## 实验过程

导入包

```
from sklearn import svm
import pprint
from collections import Counter
import jieba
import numpy as np
import pandas as pd
import csv
import os
import jieba.posseg as pseg
import sys
import string
from sklearn import feature_extraction
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from gensim import similarities
import jieba.analyse
import matplotlib.pyplot as plt
```

导入数据

```
data_wrong=pd.read_csv("features.csv")
data_right=pd.read_csv("alter_right_message_features.csv")
a=data_right.to_numpy()
b=data_wrong.to_numpy()
```

## 整理数据

```
# 将正常信息和不实信息按行合并为一个数据
x=np.concatenate((a,b),axis=0)
# 将正常信息标记为0,不实信息标记为1
y=np.array([0]*a.shape[0]+[1]*b.shape[0])
```

得到训练集，验证集，测试集

```
from sklearn import svm
from sklearn.model_selection import train_test_split

x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=1,
train_size=0.8)
x1,x2,y1,y2=train_test_split(x_test, y_test, random_state=1, train_size=0.5)
# x_train,y_train:训练集 x1,y1:验证集 x2,y2:测试集 按照8:1:1划分
```

调整参数

```
from sklearn.model_selection import GridSearchCV
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, svm
from sklearn.svm import SVC
from sklearn.datasets import make_moons, make_circles, make_classification
%matplotlib inline
#参数的范围
cc=list(np.linspace(0.1,1,10)) # 参数 C 惩罚因子
gg=list(np.linspace(0.1,1,10)) # 参数 gamma
# k折交叉验证, 这里选择了5折 (每次五份里拿一份当测试集)
grid = GridSearchCV(SVC(), param_grid={"C":cc, "gamma": gg}, cv=5)
grid.fit(x_train, y_train)

# 这里给出我们训练的最好的模型在验证集上的得分
print("验证集在最佳模型上得分为: {:.3f}".format(grid.score(x2,y2)))
# 下面给出训练最好的模型对应的参数 以及 该模型的最终得分 (百分制)
print("The best parameters are %s with a score of %0.2f%% (grid.best_params_, grid.best_score_)"
```

结果：

```
1] ▶ ▶ M1
    print("验证集在最佳模型上得分为 : {:.3f}".format(grid.score(x2,y2)))
    print("The best parameters are %s with a score of %0f%% (grid.best_params_, grid.best_score_))

验证集在最佳模型上得分为 : 0.791
The best parameters are {'C': 0.9, 'gamma': 0.4} with a score of 0.744945
```

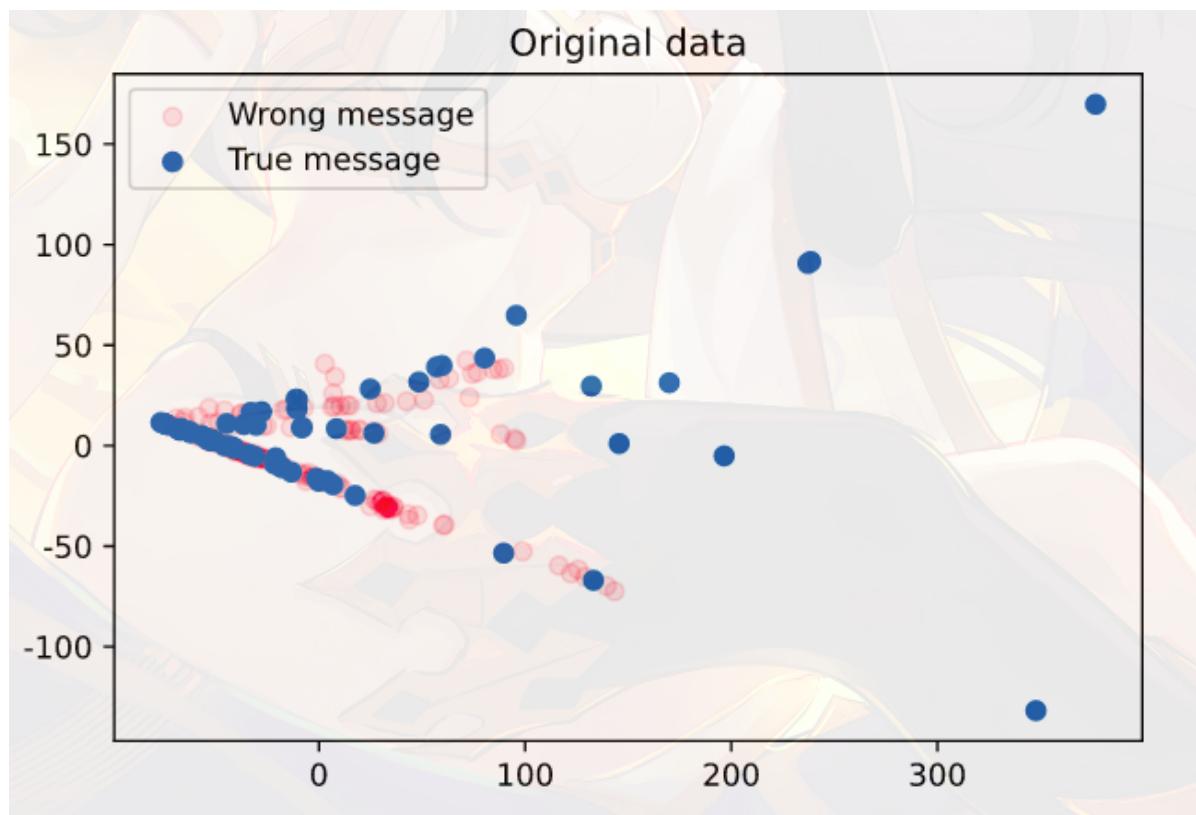
通过测试集来检验模型：

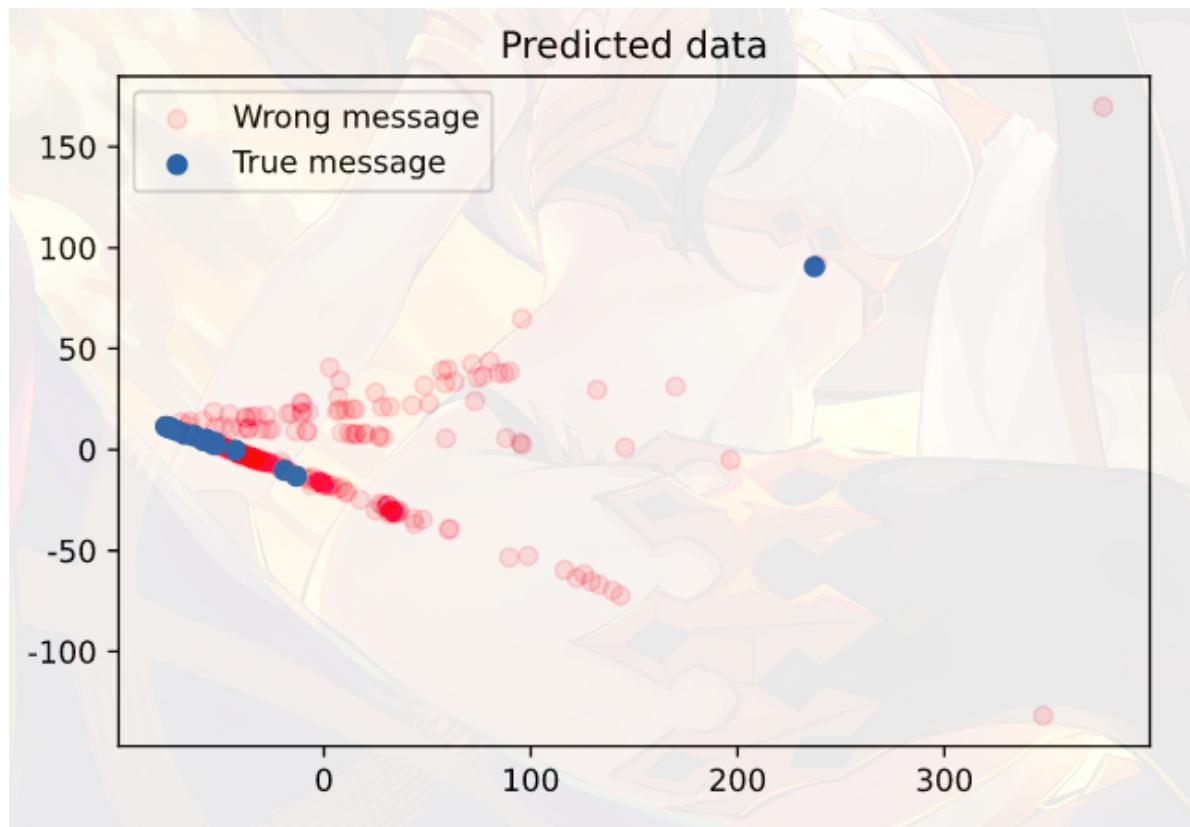
```
30] ▶ ▶ M1
    # clf = svm.SVC(C=0.1, kernel='Linear', decision_function_shape='ovr')
    model = svm.SVC(C=0.9, kernel='rbf', gamma=0.4, decision_function_shape='ovr')
    model.fit(x_train, y_train)
    print("测试集在该模型上得分为 : {:.3f}".format(round(model.score(x1, y1),3)))
    #Predict Output
    x1_predicted= model.predict(x1)
    # print(predicted)

测试集在该模型上得分为 : 0.748
```

我们想探究该模型（九维超平面）在投影到二维平面下效果如何。下面我们通过pca将所有的点强行降到二维之后，直观的感受一下我们训练的模型的好坏

- wrong message: 不实信息
- true message: 正常信息





可以看到降为降维至二维时模型表现效果很差，主要原因在于由于维数缩减严重，导致特征信息的损失量过大。但由9维特征所训练出的超平面表现效果良好，验证正确率可以达到79%，因此依然使用原模型进行分析。