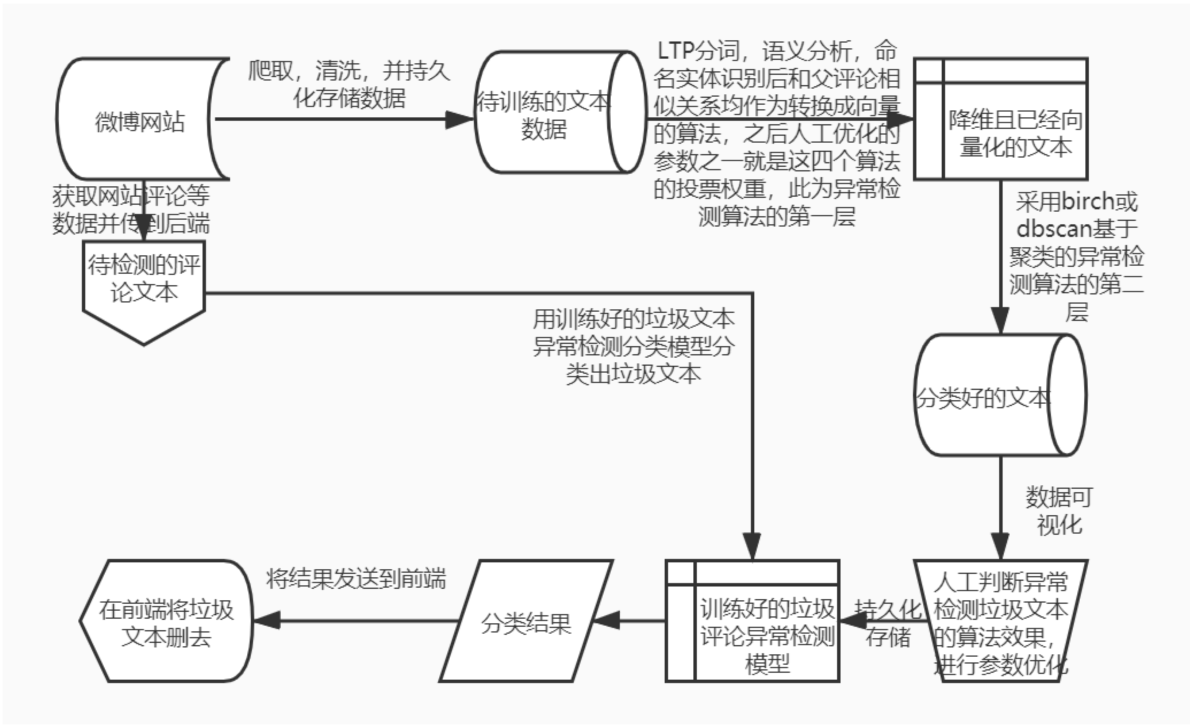
设计架构图


微博网站
爬取，清洗，并持久化存储数据
待训练的文本数据
LTP分词，语义分析，命名实体识别后和父评论相似关系均作为转换成向量的算法，之后人工优化的参数之一就是这四个算法的投票权重，此为异常检测算法的第一层
降维且已经向量化的文本
获取网站评论等数据并传到后端
待检测的评论文本
用训练好的垃圾文本异常检测分类模型分类出垃圾文本
采用birch或dbscan基于聚类的异常检测算法的第二层
分类好的文本
数据可视化
在前端将垃圾文本删去
将结果发送到前端
分类结果
训练好的垃圾评论异常检测模型
持久化存储
人工判断异常检测垃圾文本的算法效果，进行参数优化

上述内容实现以后，可以优化的地方是

- [ ] 将新拿到的文本也进行聚类，增量式聚类，双向绑定，不断优化异常检测的模型。
- [ ] 对已经分类过的文本不再用模型进行异常检测了，直接hash返回是否删除
- [ ] 当用户访问量大时，此架构失效。需要将 Redis 分出来独立作缓存，可能要用到分布式架构。已经预测过的是否是垃圾评论的数据用 Mysql 存，确保服务器内存够用

爬取，清洗，持久化存储数据

> 省略部分细节，例如对数据库的 合并，拆分 等。
>
> 不喜求喷。

利用selenium获取进入评论网页的链接

进入 https://m.weibo.cn/ 后，不断将网页向下滚动，获取动态加载的内容，利用redis的set去重，最后获得两种有用的链接。一种形如 `/detail/213113123123124`，一种是包含中文字符的链接（例如：https://m.weibo.cn/search?containerid=231522type%3D1%26t%3D10%26q%3D%23%E6%88%91%E7%9A%84%E5%B0%8F%E7%A1%AE%E5%B9%B8%23&isnewpage=1&luicode=10000011&lfid=102803），需要进入链接对应网页后点击首个评论，之后链接便转换为前者。

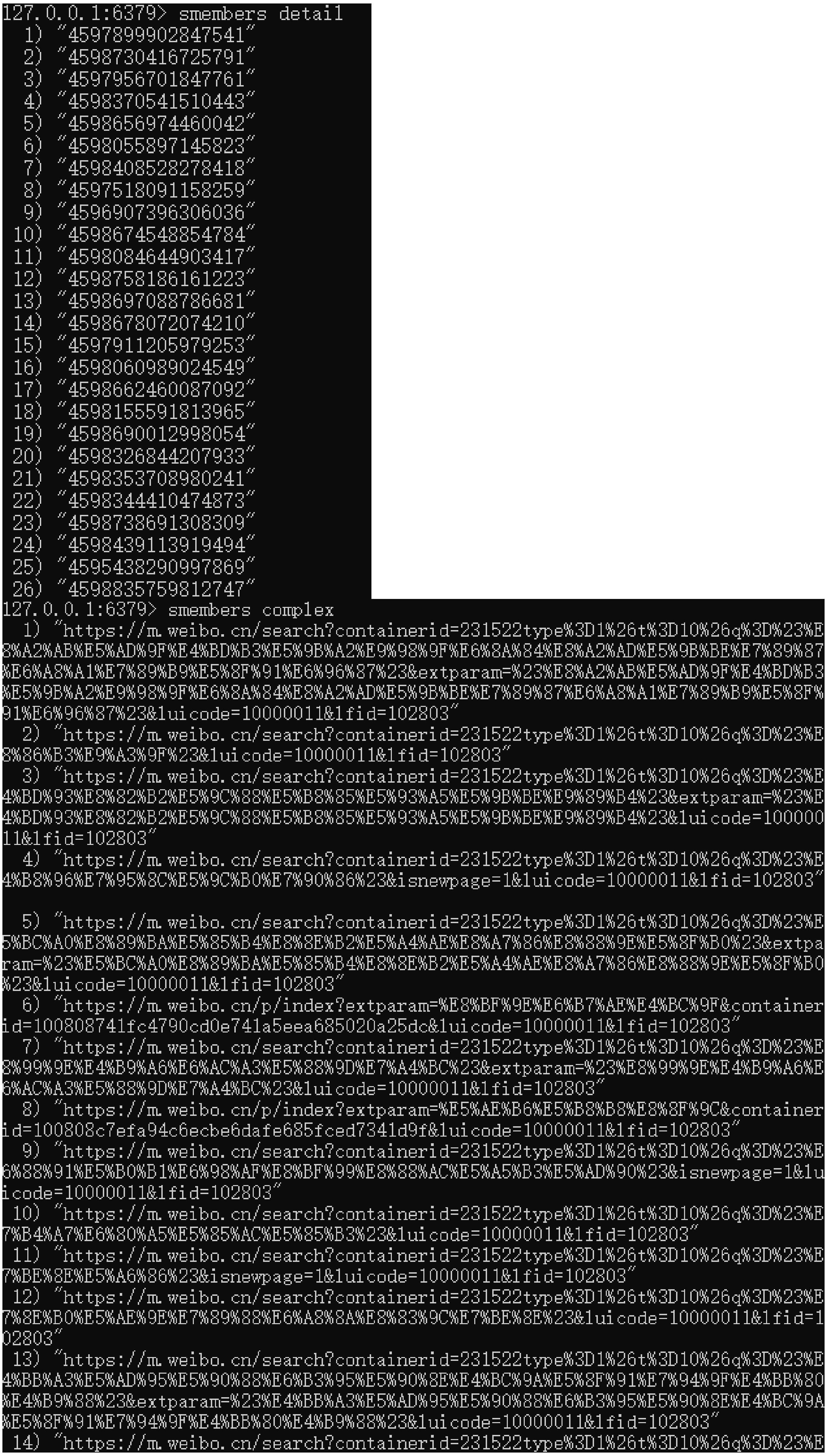
由于 链接地址从前者映射到后者 是js动态生成的，而js源码又看得我脑阔疼，也看不出。所以之后通过selenium将后者转换为前者，之后合并两份链接，统一处理。

链接获取(code)

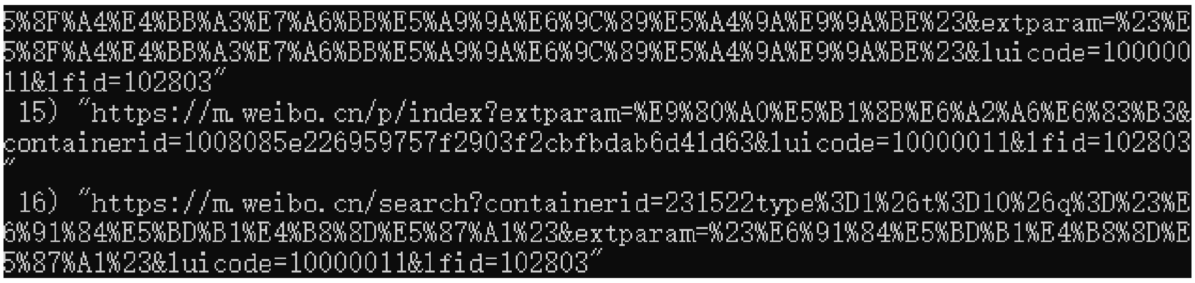
```
from selenium import webdriver
import time
import re
from lxml import etree
import redis
option = webdriver.ChromeOptions()
```

```
option.add_argument(r'--user-data-dir=D:\ChromeUserData')
wd = webdriver.Chrome(r'd:\chromedriver.exe', options=option)
wd.implicitly_wait(5)
wd.get('https://m.weibo.cn/')
wd.maximize_window()
scrapied = set()
detail_link = []
time.sleep(2)
pool = redis.ConnectionPool(host='localhost', port=6379, decode_responses=True)
r = redis.Redis(host='localhost', port=6379, decode_responses=True)
while True:
    wd.execute_script("window.scrollBy(0,3000)")
    TIMES = TIMES + 1
    html = wd.page_source
    selector = etree.HTML(html)
    list_m = selector.xpath('//*[@id="app"]/div[1]/div[2]/div[2]/div')
    print(len(list_m))
    for raw_link in list_m:
        link = raw_link.xpath('./div/div/article/div/div/div[1]/a/@href')
        if len(link) >= 1:
            link = link[0]
        else:
            link = None
        link = str(link)
        if re.match('/status', link):
            r.sadd('detail', link.split('/')[2])
        elif len(link.split('/')) > 3 and link.split('/')[2] == "m.weibo.cn":
            r.sadd('complex', link)
        else:
            print(link)
    wd.execute_script("window.scrollBy(0,6000)")
    time.sleep(2)
```

效果二瞥



```
127.0.0.1:6379> smembers detail
  1) "4597899902847541"
  2) "4598730416725791"
  3) "4597956701847761"
  4) "4598370541510443"
  5) "4598656974460042"
  6) "4598055897145823"
  7) "4598408528278418"
  8) "4597518091158259"
  9) "4596907396306036"
 10) "4598674548854784"
 11) "4598084644903417"
 12) "4598758186161223"
 13) "4598697088786681"
 14) "4598678072074210"
 15) "4597911205979253"
 16) "4598060989024549"
 17) "4598662460087092"
 18) "4598155591813965"
 19) "4598690012998054"
 20) "4598326844207933"
 21) "4598353708980241"
 22) "4598344410474873"
 23) "4598738691308309"
 24) "4598439113919494"
 25) "4595438290997869"
 26) "4598835759812747"
127.0.0.1:6379> smembers complex
  1) "https://m.weibo.cn/search?containerid=231522type%3D1%26t%3D10%26q%3D%23%E
8%A2%AB%E5%AD%9F%E4%BD%B3%E5%9B%A2%E9%98%9F%E6%8A%84%E8%A2%AD%E5%9B%BE%E7%89%87
%E6%A8%A1%E7%89%B9%E5%8F%91%E6%96%87%23&extparam=%23%E8%A2%AB%E5%AD%9F%E4%BD%B3
%E5%9B%A2%E9%98%9F%E6%8A%84%E8%A2%AD%E5%9B%BE%E7%89%87%E6%A8%A1%E7%89%B9%E5%8F%
91%E6%96%87%23&luicode=10000011&lfid=102803"
  2) "https://m.weibo.cn/search?containerid=231522type%3D1%26t%3D10%26q%3D%23%E
8%86%B3%E9%A3%9F%23&luicode=10000011&lfid=102803"
  3) "https://m.weibo.cn/search?containerid=231522type%3D1%26t%3D10%26q%3D%23%E
4%BD%93%E8%82%B2%E5%9C%88%E5%B8%85%E5%93%A5%E5%9B%BE%E9%89%B4%23&extparam=%23%E
4%BD%93%E8%82%B2%E5%9C%88%E5%B8%85%E5%93%A5%E5%9B%BE%E9%89%B4%23&luicode=100000
11&lfid=102803"
  4) "https://m.weibo.cn/search?containerid=231522type%3D1%26t%3D10%26q%3D%23%E
4%B8%96%E7%95%8C%E5%9C%B0%E7%90%86%23&isnewpage=1&luicode=10000011&lfid=102803"

  5) "https://m.weibo.cn/search?containerid=231522type%3D1%26t%3D10%26q%3D%23%E
5%BC%A0%E8%89%BA%E5%85%B4%E8%8E%B2%E5%A4%AE%E8%A7%86%E8%88%9E%E5%8F%B0%23&extpa
ram=%23%E5%BC%A0%E8%89%BA%E5%85%B4%E8%8E%B2%E5%A4%AE%E8%A7%86%E8%88%9E%E5%8F%B0
%23&luicode=10000011&lfid=102803"
  6) "https://m.weibo.cn/p/index?extparam=%E8%BF%9E%E6%B7%AE%E4%BC%9F&container
id=100808741fc4790cd0e741a5eea685020a25dc&luicode=10000011&lfid=102803"
  7) "https://m.weibo.cn/search?containerid=231522type%3D1%26t%3D10%26q%3D%23%E
8%99%9E%E4%B9%A6%E6%AC%A3%E5%88%9D%E7%A4%BC%23&extparam=%23%E8%99%9E%E4%B9%A6%E
6%AC%A3%E5%88%9D%E7%A4%BC%23&luicode=10000011&lfid=102803"
  8) "https://m.weibo.cn/p/index?extparam=%E5%AE%B6%E5%B8%B8%E8%8F%9C&container
id=100808c7efa94c6ecbe6dafe685fced7341d9f&luicode=10000011&lfid=102803"
  9) "https://m.weibo.cn/search?containerid=231522type%3D1%26t%3D10%26q%3D%23%E
6%88%91%E5%B0%B1%E6%98%AF%E8%BF%99%E8%88%AC%E5%A5%B3%E5%AD%90%23&isnewpage=1&lu
icode=10000011&lfid=102803"
 10) "https://m.weibo.cn/search?containerid=231522type%3D1%26t%3D10%26q%3D%23%E
7%B4%A7%E6%80%A5%E5%85%AC%E5%85%B3%23&luicode=10000011&lfid=102803"
 11) "https://m.weibo.cn/search?containerid=231522type%3D1%26t%3D10%26q%3D%23%E
7%BE%8E%E5%A6%86%23&isnewpage=1&luicode=10000011&lfid=102803"
 12) "https://m.weibo.cn/search?containerid=231522type%3D1%26t%3D10%26q%3D%23%E
7%8E%B0%E5%AE%9E%E7%89%88%E6%A8%8A%E8%83%9C%E7%BE%8E%23&luicode=10000011&lfid=1
02803"
 13) "https://m.weibo.cn/search?containerid=231522type%3D1%26t%3D10%26q%3D%23%E
4%BB%A3%E5%AD%95%E5%90%88%E6%B3%95%E5%90%8E%E4%BC%9A%E5%8F%91%E7%94%9F%E4%BB%80
%E4%B9%88%23&extparam=%23%E4%BB%A3%E5%AD%95%E5%90%88%E6%B3%95%E5%90%8E%E4%BC%9A
%E5%8F%91%E7%94%9F%E4%BB%80%E4%B9%88%23&luicode=10000011&lfid=102803"
 14) "https://m.weibo.cn/search?containerid=231522type%3D1%26t%3D10%26q%3D%23%E
```



```
5%8F%A4%E4%BB%A3%E7%A6%BB%E5%A9%9A%E6%9C%89%E5%A4%9A%E9%9A%BE%23&extparam=%23%E
5%8F%A4%E4%BB%A3%E7%A6%BB%E5%A9%9A%E6%9C%89%E5%A4%9A%E9%9A%BE%23&luicode=100000
11&lfid=102803"
 15) "https://m.weibo.cn/p/index?extparam=%E9%80%A0%E5%B1%8B%E6%A2%A6%E6%83%B3&
containerid=1008085e226959757f2903f2cbfbdab6d41d63&luicode=10000011&lfid=102803
"
 16) "https://m.weibo.cn/search?containerid=231522type%3D1%26t%3D10%26q%3D%23%E
6%91%84%E5%BD%B1%E4%B8%8D%E5%87%A1%23&extparam=%23%E6%91%84%E5%BD%B1%E4%B8%8D%E
5%87%A1%23&luicode=10000011&lfid=102803"
```

链接转换(code)

> 写得很丑，三个账号轮流切换，操纵js点击，数据解析，都在中间件里了，其它地方的代码不贴了，理解万岁。

```
def process_response(self, request, response, spider):
    print("spider.which_option: " + str(spider.which_option) + "  num: " +
str(spider.p_num)+ "  remain: " +
                  str(spider.r.scard("union_3_9")) + "  no_button_or_formatted:
" + str(spider.no_button))
            if spider.p_num % 34 == 0:
                print("sleeping...")
                sleep(1000)
            if spider.p_num % 10 == 0:
                print("have scrapped: " + str(spider.p_num))
                if spider.which_option == 2:
                    spider.bro.quit()
                    option2 = webdriver.ChromeOptions()
                    option2.add_argument(r'--user-data-dir=D:\ChromeUserData2')
                    spider.bro = webdriver.Chrome(r'd:\chromedriver.exe',
options=option2)
                    spider.bro.implicitly_wait(5)
                    spider.bro.get(request.url)
                    spider.which_option = 1
                elif spider.which_option == 1:
                    spider.bro.quit()
                    option1 = webdriver.ChromeOptions()
                    option1.add_argument(r'--user-data-dir=D:\ChromeUserData1')
                    spider.bro = webdriver.Chrome(r'd:\chromedriver.exe',
options=option1)
                    spider.bro.implicitly_wait(5)
                    spider.bro.get(request.url)
                    spider.which_option = 0
                else:
                    spider.bro.quit()
                    option = webdriver.ChromeOptions()
                    option.add_argument(r'--user-data-dir=D:\ChromeUserData')
                    spider.bro = webdriver.Chrome(r'd:\chromedriver.exe',
options=option)
                    spider.bro.implicitly_wait(5)
                    spider.bro.get(request.url)
                    spider.which_option = 2
            else:
                spider.bro.get(request.url)
            sleep(2)
            try:
                button = spider.bro.find_element_by_xpath(
                    '//*[@id="app"]/div[1]/div[1]/div[4]//div/footer/div[2]/i |
//*[@id="app"]/div[1]/div[1]/div[5]//div/footer/div[2]/i')
```

```
            except:
                button = None
            if button:
                spider.bro.execute_script('arguments[0].click();', button)
            else:
                try:

 print(spider.bro.find_element_by_xpath('/html/body/div/p/text()').extract_first
())
                    exit()
                except:
                    print("no button")
                    spider.r.srem(spider.db_c, request.url)
                    spider.r.sadd('no_button', request.url)
                    spider.no_button = spider.no_button + 1
                    return response
            sleep(2)
            spider.p_num = spider.p_num + 1
            link = str(spider.bro.current_url)
            ll = link.split('/')
            if len(ll) >= 5 and ll[3] == 'detail':
                spider.r.sadd('c_detail', ll[-1])
                spider.r.srem(spider.db_c, request.url)
                print("spider.bro.current_url: " + link)
            else:
                # 这里直接从数据库中删掉
                spider.r.srem(spider.db_c, request.url)
                spider.no_button = spider.no_button + 1
                print(link)
            return response
```

效果不瞥

不瞥

获取评论

以逸待劳

> 没钱，没IP，没账号，~~没脑子，~~只能sleep。。。
>
> 所谓`sleep`，就是电脑跑，我躺。

因为微博爬的稍微多一点就会被限制，然后就要等个10到20分钟才能继续爬，所以scrapy那个异步的玩意儿没用，异到一半给你个403，然后你还得保留一群没异完的数据来知道自己上次爬到哪儿了，代码量杠杠的，头发萧萧的，效率提升可怜，还不如给爷一个个链接爬，在数据解析那一块直接持久化存储，爬了遇到不是200的，直接一个sleep。当然，可以切换个cookie，或搞两个代理ip试试，如果搞代理的钱给报销就好了，可惜报不得。。爬到结尾遇到ok=0了，说明一条数据完了，把爬过的id在数据库中删掉，相当于保存状态，然后继续下一条。不让它异步，把CONCURRENT_REQUESTS设置为1，感觉拿个框架搞这个有点像高射炮打蚊子。微博需要手机验证码登录真的恶心(比淘宝拿深度学习搞稍微好点)，虽然之后搞个cookie可以解决，鬼想去看源码分析cookie中有没有键是通过时间来验证的，就像有道字典网页版的cookie一样，虽然直接复制cookie在之后爬【被评论的文本】确实有效果。

微博账号|手机号 少，代理IP仅限翻墙用的VPN，我甚至想 搞个 多个热点、一个wifi、一个VPN 自动切换的脚本，但理智与对自己精力与实力清醒的认识让我放弃了这个冲动。

人生不易，躺平：以逸待劳

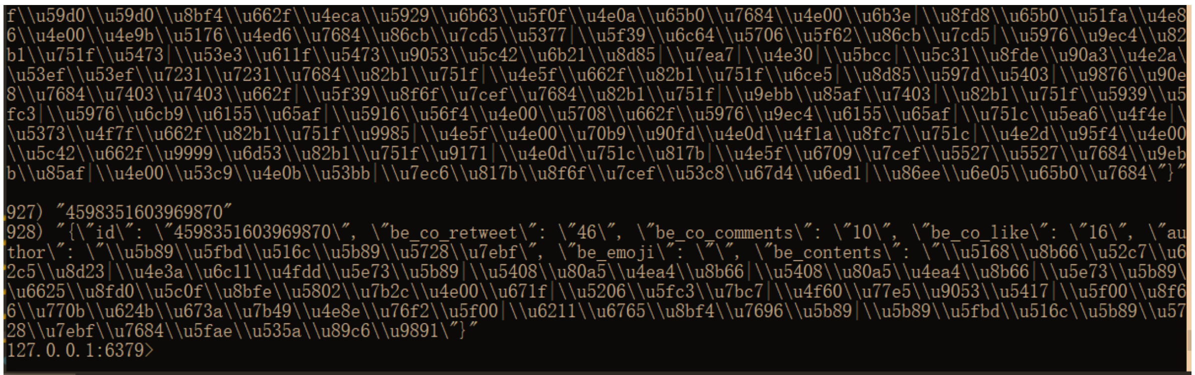
爬取清洗并持久化存储被评论的文本信息(code)

> 开始用 requests 做试验，然后 scrapy 实现，结果同样的思路，scrapy 中除了多了这个场景不需要的异步，还出现了一个很诡异的问题，估计是我 cookie 设置没有生效，爬到一半又是弹出登录界面，不过这个报错我 requests 中没遇到过，暂且将这个错误情况收入囊中，给之后的 requests 用。scrapy 框架是死的，requests 库是活的，所以 评论的爬取与持久化存储 用 requests 了。
>
> 写这玩意儿花了我将近一天时间，呜~ 数据量越大，边界条件越复杂，真是【if else try catch 地狱】啊！

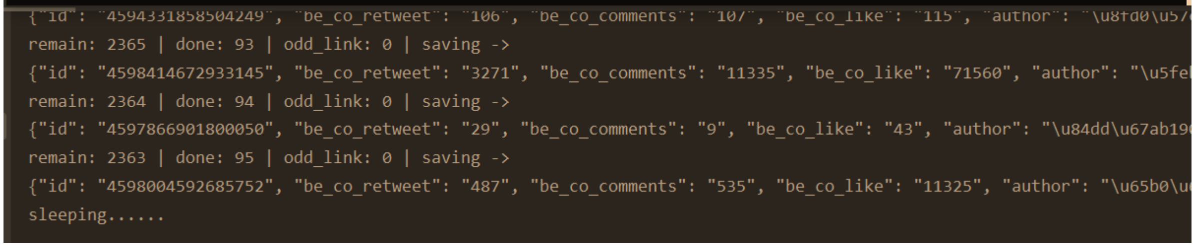
```
import requests
import re
import redis
import json
from time import sleep
db_key = 'copy_all'
headers = {
"user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/88.0.4324.104 Safari/537.36",
"cookie": "WEIBOCN_FROM=1110006030; SUB=_2A25NEVOtDeRhGeBL7lYY9ybFyDiIHXVu-
mNlrDV6PUJbkdAKLXbwkW1NRutLlmTowwVwMFvx2VGceNikaFWGhsY8; MLOGIN=1;
_T_WM=67795908980;
M_WEIBOCN_PARAMS=oid%3D4598439113919494%26luicode%3D20000061%26lfid%3D4598439113
919494%26uicode%3D20000061%26fid%3D4598439113919494; XSRF-TOKEN=657a72"
}
be_cf_url = "https://m.weibo.cn/detail/{}"
pool = redis.ConnectionPool(host='localhost', port=6379, decode_responses=True)
r = redis.Redis(host='localhost', port=6379, decode_responses=True)
id_pool = r.smembers(db_key)
continous = 0
cannot_sum = 0
interval_controller = 1
headertimes = 1
for u_id in id_pool:
    interval_controller += 1
    if interval_controller % 97 == 0 or headertimes % 11 == 0:
        print("sleeping......")
        headertimes = 1
        interval_controller += 1
        sleep(1000)
    continous = 0
    be_c_url = be_cf_url.format(u_id)
    be_res = requests.get(url=be_c_url)
    if be_res.content and be_res.status_code == 200:
        bec_l = re.split(r'"text":|"textLength":|"source":', be_res.text)
        if len(bec_l) < 2:
            print('now that I use headers')
            headertimes += 1
            be_res = requests.get(url=be_c_url, headers=headers)
            if be_res.content and be_res.status_code == 200:
                bec_l = re.split(r'"text":|"textLength":|"source":',
be_res.text)
            else:
                print("how come???  " + be_c_url)
                exit()
            if len(bec_l) < 2:
                print("napping...")
                print("odd: " + be_c_url)
                r.sadd("odd_link", u_id)
                with open('assert_login_required.html', 'w+', encoding='utf-8')
as f:
                    f.write(be_res.text)
                sleep(4)
                continue
        else:
            be_emoji_contents = bec_l[1]  # 1 big
            be_emoji = re.findall(r'alt=\[(.+?)\]+', be_emoji_contents)
            be_contents = re.findall(r'[\u4e00-\u9fa5]+', be_emoji_contents)
            author = re.split(r'"profile_image_url":|"screen_name":',
be_res.text)[1].strip()[1: -2]
            be_co_list =
re.split(r'"reposts_count":|"comments_count":|"attitudes_count":|"pending_approv
al_count":', be_res.text)
            be_co_retweet = re.findall(r'\d+', be_co_list[-4])[0]
            be_co_comments = re.findall(r'\d+', be_co_list[-3])[0]
            be_co_like = re.findall(r'\d+', be_co_list[-2])[0]
        be_emoji_ser = '|'.join(be_emoji)
        be_contents_ser = '|'.join(be_contents)
        obj = {"id": u_id, "be_co_retweet": be_co_retweet, "be_co_comments":
be_co_comments,
               "be_co_like": be_co_like, "author": author, "be_emoji":
be_emoji_ser, "be_contents": be_contents_ser}
        value = json.dumps(obj)
        if r.hset("vb_article", u_id, value):
            print("remain: " + str(r.scard(db_key)) + " | done: " +
str(interval_controller - 1) + " | odd_link: " + str(r.scard("odd_link")) + " |
saving -> ")
            print(value)
        else:
            print("remain: " + str(r.scard(db_key)) + " | done: " +
str(interval_controller - 1) + " | odd_link: " + str(r.scard("odd_link")) + " |
duplicated -> ")
            print(value)
        r.srem(db_key, u_id)
    else:
        try:
            print(be_res.text)
            print("的确把我禁了，呜——")
        except:
            print("WTF?!")
        print(be_res)
        exit()
```

效果一瞥

> 还在爬，先瞥了。



```
f\\u59d0\\u59d0\\u8bf4\\u662f\\u4eca\\u5929\\u6b63\\u5f0f\\u4e0a\\u65b0\\u7684\\u4e00\\u6b3e|\\u8fd8\\u65b0\\u51fa\\u4e8
6\\u4e00\\u4e9b\\u5176\\u4ed6\\u7684\\u86cb\\u7cd5\\u5377|\\u5f39\\u6c64\\u5706\\u5f62\\u86cb\\u7cd5|\\u5976\\u9ec4\\u82
b1\\u751f\\u5473|\\u53e3\\u611f\\u5473\\u9053\\u5c42\\u6b21\\u8d85|\\u7ea7|\\u4e30|\\u5bcc|\\u5c31\\u8fde\\u90a3\\u4e2a\
\u53ef\\u53ef\\u7231\\u7231\\u7684\\u82b1\\u751f|\\u4e5f\\u662f\\u82b1\\u751f\\u6ce5|\\u8d85\\u597d\\u5403|\\u9876\\u90e
8\\u7684\\u7403\\u7403\\u662f|\\u5f39\\u8f6f\\u7cef\\u7684\\u82b1\\u751f|\\u9ebb\\u85af\\u7403|\\u82b1\\u751f\\u5939\\u5
fc3|\\u5976\\u6cb9\\u6155\\u65af|\\u5916\\u56f4\\u4e00\\u5708\\u662f\\u5976\\u9ec4\\u6155\\u65af|\\u751c\\u5ea6\\u4f4e|\
\u5373\\u4f7f\\u662f\\u82b1\\u751f\\u9985|\\u4e5f\\u4e00\\u70b9\\u90fd\\u4e0d\\u4f1a\\u8fc7\\u751c|\\u4e2d\\u95f4\\u4e00
\\u5c42\\u662f\\u9999\\u6d53\\u82b1\\u751f\\u9171|\\u4e0d\\u751c\\u817b|\\u4e5f\\u6709\\u7cef\\u5527\\u5527\\u7684\\u9eb
b\\u85af|\\u4e00\\u53c9\\u4e0b\\u53bb|\\u7ec6\\u817b\\u8f6f\\u7cef\\u53c8\\u67d4\\u6ed1|\\u86ee\\u6e05\\u65b0\\u7684\"}"

927) "4598351603969870"
928) "{\"id\": \"4598351603969870\", \"be_co_retweet\": \"46\", \"be_co_comments\": \"10\", \"be_co_like\": \"16\", \"au
thor\": \"\\u5b89\\u5fbd\\u516c\\u5b89\\u5728\\u7ebf\", \"be_emoji\": \"\", \"be_contents\": \"\\u5168\\u8b66\\u52c7\\u6
2c5\\u8d23|\\u4e3a\\u6c11\\u4fdd\\u5e73\\u5b89|\\u5408\\u80a5\\u4ea4\\u8b66|\\u5408\\u80a5\\u4ea4\\u8b66|\\u5e73\\u5b89\
\u6625\\u8fd0\\u5c0f\\u8bfe\\u5802\\u7b2c\\u4e00\\u671f|\\u5206\\u5fc3\\u7bc7|\\u4f60\\u77e5\\u9053\\u5417|\\u5f00\\u8f6
6\\u770b\\u624b\\u673a\\u7b49\\u4e8e\\u76f2\\u5f00|\\u6211\\u6765\\u8bf4\\u7696\\u5b89|\\u5b89\\u5fbd\\u516c\\u5b89\\u57
28\\u7ebf\\u7684\\u5fae\\u535a\\u89c6\\u9891\"}"
127.0.0.1:6379>
```



```
{"id": "4594331858504249", "be_co_retweet": "106", "be_co_comments": "107", "be_co_like": "115", "author": "\u8fd0\u57
remain: 2365 | done: 93 | odd_link: 0 | saving ->
{"id": "4598414672933145", "be_co_retweet": "3271", "be_co_comments": "11335", "be_co_like": "71560", "author": "\u5fe
remain: 2364 | done: 94 | odd_link: 0 | saving ->
{"id": "4597866901800050", "be_co_retweet": "29", "be_co_comments": "9", "be_co_like": "43", "author": "\u84dd\u67ab19
remain: 2363 | done: 95 | odd_link: 0 | saving ->
{"id": "4598004592685752", "be_co_retweet": "487", "be_co_comments": "535", "be_co_like": "11325", "author": "\u65b0\u
sleeping......
```

爬取并持久化存储评论别人的文本信息(code)

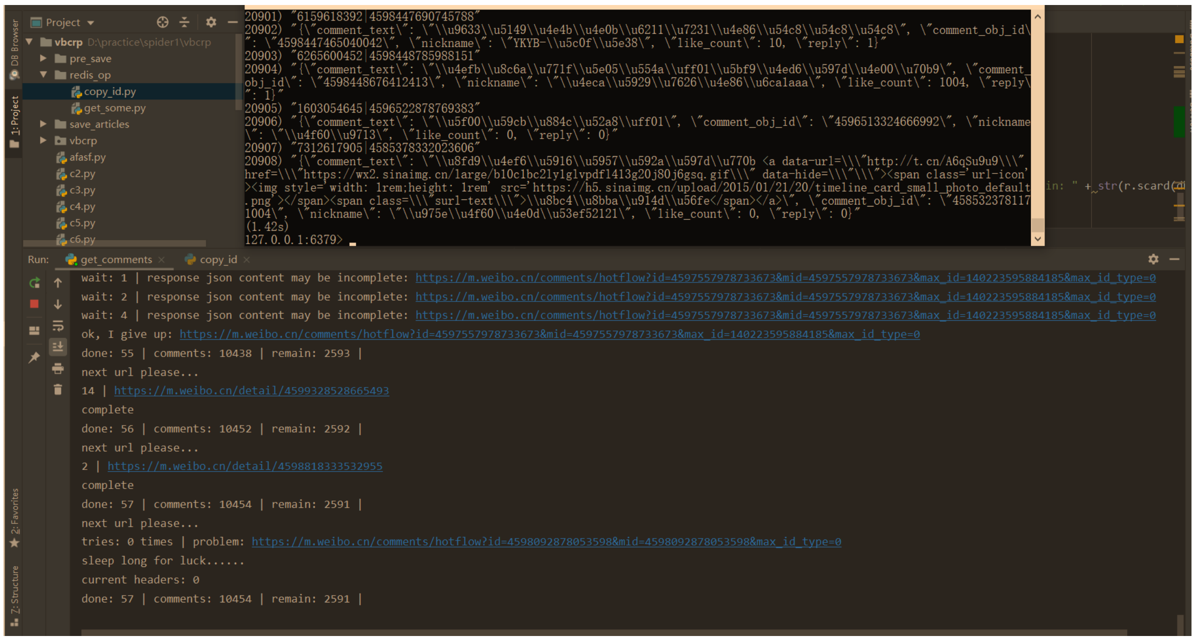
> 这个数据量更大，花半天写完后，就让它跑了。测试了总共有100个id左右，爬了两万多条评论，感觉可以了，就开始漫长的跑爬虫了。不出意外的话，估计跑1天可以拿到40万条评论。让我们拭目以待。
>
> 这玩意儿跑这么长时间只能拿到这么一点数据主要是我账号太少了，只有3个。让我不得不在代码中精打细算，还好没加入保存长得号对应的状态，我试着写过，写到200多行就不敢继续写了，到时候debug不知得花多久时间，不如直接进行下一个链接，把上一个可能爬到一半的放到最后再去爬。
>
> 如果给我的账号数量增加1000倍，也就是给我3000个cookie和UA组（没看源码，估计它的加密涉及UA，所以UA和cookie要对应），增量式，把微博的评论一直爬完，，，不是问题。可惜有不得。

爬到20多万条数据时，出现了证书出错的情况。这个错误给我的感觉是随机发生的。代码有改动。

```
import requests
import json
from time import sleep
import redis
# import urllib3
# urllib3.disable_warnings(urllib3.exceptions.InsecureRequestWarning)


def dialog(n, t, l, m, c):
    with open('{level}_dialog_{name}.{type}'.format(name=n, type=t, level=l),
'{method}'.format(method=m),
              encoding='utf-8') as f:
        f.write(c)


pool = redis.ConnectionPool(host='localhost', port=6379, decode_responses=True)
r = redis.Redis(host='localhost', port=6379, decode_responses=True)
db_id = "copy_all"
out = "out4"
db_store_comments = "comment_raw"
headers = [
    {
        "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.96 Safari/537.36",
        "cookie": "WEIBOCN_FROM=1110006030;
SUB=_2A25NEV0tDeRhGeBL7lYY9ybFyDiIHXVu-
mNlrDV6PUJbkdAKLXbwkW1NRutLlmTowwVwMFvx2VGceNikaFWGhsY8; MLOGIN=1;
_T_WM=67795908980;
M_WEIBOCN_PARAMS=oid%3D4598812490338528%26luicode%3D20000061%26lfid%3D4598812490
338528; XSRF-TOKEN=6ccdfa"
    },
    {
        "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.104 Safari/537.36",
        "cookie":
"SUB=_2A25NCGyZDeRhGeFL7lYW8ifKyjuIHXVu83TRrDV6PUJbkdAKLXftkW1NfedyhZeSJiB2FIIkB
xvJsYliT6pqVB5r; _T_WM=51270776894; MLOGIN=1; WEIBOCN_FROM=1110006030;
M_WEIBOCN_PARAMS=luicode%3D10000011%26lfid%3D102803%26uicode%3D10000011%26fid%3D
102803; XSRF-TOKEN=7b2f74"
    },
    {
        "user-agent": "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.104 Safari/537.36",
        "cookie":
"SUB=_2A25NEfqWDeRhGeFN61EZ8C3FzzuIHXVu_YberDV6PUJbkdANLWelkW1NQJzKAyjB5YSsbaRBC
twFYN7bdNCHez3d; _T_WM=99361324838; XSRF-TOKEN=2da516; WEIBOCN_FROM=1110006030;
MLOGIN=1; M_WEIBOCN_PARAMS=luicode%3D20000174%26uicode%3D20000174"
    }
]
common_url = "https://m.weibo.cn/detail/{}"
start_get_uormat = "https://m.weibo.cn/comments/hotflow?id={id}&mid=
{id}&max_id_type=0"
next_ajax_uormat = "https://m.weibo.cn/comments/hotflow?id={id}&mid={id}&max_id=
{max_id}&max_id_type={max_id_type}"
url_pool = list(r.smembers(db_id))
next_url_flag = False
hi = 1
luck = 0
for url_id in url_pool:
    cur_url = start_get_uormat.format(id=url_id)
    break_big = False
    try:
        res = requests.get(url=cur_url, headers=headers[hi])  #, verify=False
    except:
        print("证书失效1")
        break_big = True
    cnum = 0
    exp = 0
    next_url = None
    while True:
        if break_big:
            break
        tries = 0
        sp = 0
        # turn = 0
        # get res_text or give up
        while tries < 3 and sp < 5:
            '''
                sp: 403 || 200 but have not got content
                tries: comments load complete or I give up
            '''
            if res.status_code == 200 and res.content:  # 分开?
                try:
                    exp = 0
                    res_text = json.loads(res.text)
                except:
                    if next_url:
                        print("maybe login required and need to change cookie or
just sleep " + next_url)
                    else:
                        print("maybe login required and need to change cookie or
just sleep " + cur_url)
                    dialog(n=url_id, m='w+', l='f_error', c=res.text, t='html')
                    print(hi)
                    tries += 1
                    exp += 1
                    sleep(2 ** (tries - 1))
                    if exp > 3:
                        exp = 0
                        print("long sleep...............")
                        sleep(1000)
                    # exit()
                if str(res_text['ok']) == "0":
                    tt = 2 ** tries
                    sleep(tt)
                    tries += 1
                    try:
                        if next_url:
                            print("wait: " + str(tt) + " | " + "response json
content may be incomplete: " + next_url)
                            res = requests.get(url=next_url,
headers=headers[hi])  #, verify=False
                        else:
                            print("wait: " + str(tt) + " | " + "response json
content may be incomplete: " + cur_url)
                            res = requests.get(url=cur_url, headers=headers[hi])
 #, verify=False
                    except:
                        print("证书失效2")
                        break_big = True
                        break
                else:
                    break
            else:  # 这种情况大概率是请求频繁了
                if next_url:
                    print("tries: " + str(tries) + " times | problem: " +
next_url)
                else:
                    print("tries: " + str(tries) + " times | problem: " +
cur_url)
                luck += 1
                if luck <= 2:
                    print("snap a luck, drop header: " + str(hi))
                    sleep(2 ** (luck - 1))
                elif luck < len(headers) + 2:
                    # turn += 1
                    hi += 1
                    hi %= len(headers)
                    break_big = True
                    break
                else:
                    print("sleep long for luck......")
                    print("current headers: " + str(hi))
                    print("done: " + str(2648 - r.scard(db_id)) + " | comments: 
" + str(
                        r.hlen(db_store_comments)) + " | remain: " + 
str(r.scard(db_id)) + " | ")
                    luck = 0
                    sleep(1000 - len(headers))
                    if sp > 4:
                        print("这还请求频繁? 小喇叭! ")
                        print(hi)
                        print("done: " + str(2648 - r.scard(db_id)) + " | 
comments: " + str(
                            r.hlen(db_store_comments)) + " | remain: " + 
str(r.scard(db_id)) + " | ")
                        exit()
                    sp += 1
        else:
            if next_url:
                print("ok, I give up: " + next_url)
            else:
                print("ok, I give up: " + cur_url)
            r.srem(db_id, url_id)
            break
        if break_big:
            break
        try:
            data = res_text["data"]
            data_list = data["data"]
        except:
            dialog(n=url_id, m='w+', l='u_error', c=res.text, t='html')
            print("unexpected response: " + next_url)
            print(hi)
            exit()
        for comment in data_list:
            user_id = comment['user']['id']
            comment_id = comment['id']
            comment_key = str(user_id) + "|" + comment_id
            obj = {
                "comment_text": comment['text'],
                "comment_obj_id": url_id,
                "nickname": comment['user']['screen_name'],
                "like_count": comment['like_count'],
                "reply": comment['total_number']
            }
            str_obj = json.dumps(obj)
            dialog(n=out, l="print", t="txt", m="a+", c=comment_key + '\n' + 
str_obj + '\n\n')
            r.hset(db_store_comments, comment_key, str_obj)
        cnum += len(data_list)
        print(str(cnum) + " | " + common_url.format(url_id))
        max_id = data["max_id"]
        if str(max_id) == "0":
            print("complete")
            r.srem(db_id, url_id)
            break
        else:
            next_url = next_ajax_uormat.format(id=url_id, max_id=max_id,
max_id_type=data["max_id_type"])
            try:
                res = requests.get(url=next_url, headers=headers[hi])  #,
verify=False
            except:
                print("证书失效3")
                break_big = True
                break
    if break_big:
        print("regardless status, new link please")
        sleep(4)
        url_pool.append(url_id)
    else:
        print("done: " + str(2648 - r.scard(db_id)) + " | comments: " +
str(r.hlen(db_store_comments)) + " | remain: " + str(r.scard(db_id)) + " | ")
        print("next url please...")
```

效果半瞥

> 先~~忍不住~~放爬了1小时左右时候的截图。



```
20901) "6159618392|4598447690745788"
20902) "{\"comment_text\": \"\\u9633\\u5149\\u4e4b\\u4e0b\\u6211\\u7231\\u4e86\\u54c8\\u54c8\\u54c8\", \"comment_obj_id\
": \"4598447465040042\", \"nickname\": \"YKYB-\\u5c0f\\u5e38\", \"like_count\": 10, \"reply\": 1}"
20903) "6265600452|4598448785988151"
20904) "{\"comment_text\": \"\\u4efb\\u8c6a\\u771f\\u5e05\\u554a\\uff01\\u5bf9\\u4ed6\\u597d\\u4e00\\u70b9\", \"comment_
obj_id\": \"4598448676412413\", \"nickname\": \"\\u4eca\\u5929\\u7626\\u4e86\\u6ca1aaa\", \"like_count\": 1004, \"reply\
": 1}"
20905) "1603054645|4596522878769383"
20906) "{\"comment_text\": \"\\u5f00\\u59cb\\u884c\\u52a8\\uff01\", \"comment_obj_id\": \"4596513324666992\", \"nickname
\": \"\\u4f60\\u9713\", \"like_count\": 0, \"reply\": 0}"
20907) "7312617905|4585378332023606"
20908) "{\"comment_text\": \"\\u8fd9\\u4ef6\\u5916\\u5957\\u592a\\u597d\\u770b <a data-url=\\\"http://t.cn/A6qSu9u9\\\"
href=\\\"https://wx2.sinaimg.cn/large/b10c1bc2ly1glvpdf1413g20j80j6gsq.gif\\\" data-hide=\\\"\\\">< span class='url-icon'
><img style='width: 1rem;height: 1rem' src='https://h5.sinaimg.cn/upload/2015/01/21/20/timeline_card_small_photo_default
.png'></span><span class=\\\"surl-text\\\">\\u8bc4\\u8bba\\u914d\\u56fe</span></a>\", \"comment_obj_id\": \"458532378117
1004\", \"nickname\": \"\\u975e\\u4f60\\u4e0d\\u53ef52121\", \"like_count\": 0, \"reply\": 0}"
(1.42s)
127.0.0.1:6379> 
```

```
wait: 1 | response json content may be incomplete: https://m.weibo.cn/comments/hotflow?id=4597557978733673&mid=4597557978733673&max_id=140223595884185&max_id_type=0
wait: 2 | response json content may be incomplete: https://m.weibo.cn/comments/hotflow?id=4597557978733673&mid=4597557978733673&max_id=140223595884185&max_id_type=0
wait: 4 | response json content may be incomplete: https://m.weibo.cn/comments/hotflow?id=4597557978733673&mid=4597557978733673&max_id=140223595884185&max_id_type=0
ok, I give up: https://m.weibo.cn/comments/hotflow?id=4597557978733673&mid=4597557978733673&max_id=140223595884185&max_id_type=0
done: 55 | comments: 10438 | remain: 2593 |
next url please...
14 | https://m.weibo.cn/detail/4599328528665493
complete
done: 56 | comments: 10452 | remain: 2592 |
next url please...
2 | https://m.weibo.cn/detail/4598818333532955
complete
done: 57 | comments: 10454 | remain: 2591 |
next url please...
tries: 0 times | problem: https://m.weibo.cn/comments/hotflow?id=4598092878053598&mid=4598092878053598&max_id_type=0
sleep long for luck......
current headers: 0
done: 57 | comments: 10454 | remain: 2591 |
```

数据处理

数据库设计

数据项(data item)

> 以hash形式储存在redis中，以前三个id的组合为key。

key	value
comment_id	该条评论的 ID
user_id	发出该条评论的 用户ID
url_id	该评论评论对象 对应的 网页ID
comment_text	该评论的 所有中文文本，包括表情指代词
comment_emoji	该评论的 表情指代词(依次顺序出现在comment_text中)
nickname	该评论作者 在发出这条评论时所用昵称
like_count	该条评论的 被 点赞的次数
reply_count	该条评论的 被 回复的次数
be_co_retweet	该评论评论对象 被 转发的次数
be_co_comments	该评论评论对象 拥有的评论量
be_co_like	该评论评论对象 被点赞的次数
author	该评论评论对象的 用户名
be_contents	该评论评论对象的 所有中文文本
be_emoji	该评论评论对象的 表情指代词(依次顺序出现在comment_text中)

数据库连接与存储(code)

> 实现上述模型

```
import redis
import re
import json


def str_to_json_articles(article_str):
    o_article = json.loads(article_str)
    be_contents = o_article['be_contents'].split('|')
    be_emoji = o_article['be_emoji'].split('|')
    o_article['be_contents'] = be_contents
    o_article['be_emoji'] = be_emoji
    return o_article


def str_to_json_comments(comments_str):
    o_comment = json.loads(comments_str)
    comment = o_comment['comment_text']
    comment_emoji = re.findall(r'alt=\[(.*?)\]', comment)
    comment_text = re.findall(r'[\u4e00-\u9fa5]+', comment)
    o_comment['comment_text'] = comment_text
    o_comment['comment_emoji'] = comment_emoji
    return o_comment


def obj_to_string(ots):
    r_o = {}
    for key, value in ots.items():
        r_o[key] = '|'.join(value) if isinstance(value, list) else value
    # print(r_o)
    return json.dumps(r_o)


pool = redis.ConnectionPool(host='localhost', port=6379, decode_responses=True)
r = redis.Redis(host='localhost', port=6379, decode_responses=True)

read_articles = 'vb_article'
read_comments = 'comment_raw'
write_comments = 'comments_zh'

t_hash = r.hgetall(read_comments)
articles = r.hgetall(read_articles)
for key, value in t_hash.items():
    [user_id, comment_id] = key.split('|')
    comment_obj = str_to_json_comments(value)
    url_id = comment_obj['comment_obj_id']
    articles_str = articles[str(url_id)]
    articles_obj = str_to_json_articles(articles_str)
    obj = {
        "comment_text": comment_obj["comment_text"],
        "comment_emoji": comment_obj["comment_emoji"],
        "nickname": comment_obj["nickname"],
        "like_count": comment_obj["like_count"],
        "reply_count": comment_obj["reply"],
        "be_co_retweet": articles_obj["be_co_retweet"],
        "be_co_comments": articles_obj["be_co_comments"],
        "be_co_like": articles_obj["be_co_like"],
        "author": articles_obj["author"],
        "be_contents": articles_obj["be_contents"],
        "be_emoji": articles_obj["be_emoji"]
    }
    s_keys = '|'.join([comment_id, user_id, comment_obj["comment_obj_id"]])
    s_value = obj_to_string(obj)
    r.hset(write_comments, s_keys, s_value)
```

异常检测算法第一层

算法选择

已复现网络与论文上的算法评测代码，了解无监督聚类异常检测算法的原理与发展情况，核心代码和实现可参考我的读书笔记。

划分任务(弃用)

> 提需求，做自己的产品经理

- [x] 将 Redis 中所有文本分词后添加到语料库
- [x] 将 Redis 中文本分词，加载停用词表
- [x] 语料库向量化，if-idf 赋权

- [x] 配好环境(Docker 微服务架构)，在ubuntu上跑通前面所有的东西
- [ ] PCA降维，dbscan 聚类(可拓展选择其它算法)
- [ ] 输出(可能要修改前面的传参)

实现需求

> 语料库要尽量包含所有的词
>
> 先用小数据集模拟，避免大数据集运行时间过长

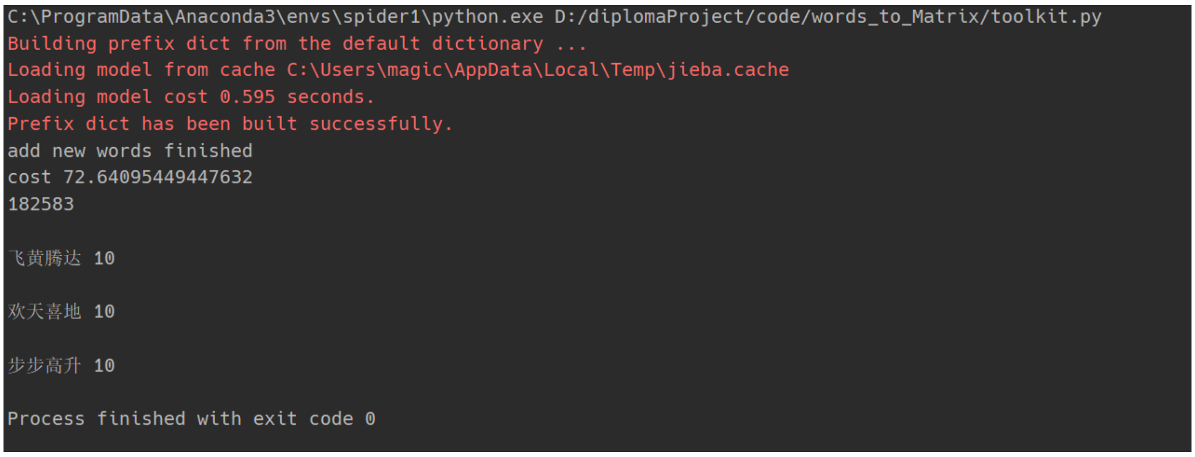
```
import redis
import json
import jieba
import jieba.analyse
import time
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import PCA


corpus_file = "idf.txt.big"  # "idf.txt.big"
stop_words = open("stopwords.txt", encoding='utf-8').read()
corpus_fp = open(corpus_file, 'a+', encoding='utf-8')
raw_corpus = open(corpus_file, encoding='utf-8').readlines()
topK = 10


def remove_stop_words(data_list):
    word_list = []
    for data in data_list:
        if data not in stop_words:
            word_list.append(data)
    return word_list


def preheat_cache(db, sensitivity=4):
    """
    preheat 分词引擎
    add new words if discovered from comments,
    everytime service started, traversal the database, add new words to cache
    takes long time...
    wish astringency
    """
    pool = redis.ConnectionPool(host='localhost', port=6379,
decode_responses=True)
    r = redis.Redis(host='localhost', port=6379, decode_responses=True)
    jieba.analyse.set_idf_path(corpus_file)
    raw_comment = r.hgetall(db)
    raw_new_words_hash_map = {}
    start_time1 = time.time()
    for key, raw_value in raw_comment.items():
        value = json.loads(raw_value)
        comment_text = value['comment_text']
        comment_array = comment_text.split('|')
        comment = []
        for text in comment_array:
            # comment.append('/'.join(jieba.cut(text)))
            comment.append('/'.join(jieba.analyse.extract_tags(text,
topK=topK)))
        comment_row = '/'.join(comment)
        no_stop_words = remove_stop_words(comment_row.split('/'))
        for word in no_stop_words:
            if word in raw_new_words_hash_map:
                raw_new_words_hash_map[word] = raw_new_words_hash_map[word] + 1
            else:
                raw_new_words_hash_map[word] = 0
            jieba.add_word(word)
    print('add new words finished')
    new_words_hash_map = {}
    for key_word, times in raw_new_words_hash_map.items():
        if times >= sensitivity:
            new_words_hash_map[key_word] = times
    end_time1 = time.time()
    print("cost " + str(end_time1 - start_time1))
    append_corpus(new_words_hash_map)


def append_corpus(new_words):
    corpus = set()
    size = len(new_words)
    for item in raw_corpus:
        w = item.split(' ')[0]
        corpus.add(w)
    print(len(corpus))
    for key_word, times in new_words.items():
        if key_word not in corpus and len(key_word) > 2:
            new_item = '\n' + key_word + " " + str(topK)  # str(times/size)
            print(new_item)
            corpus_fp.write(new_item)


def text_to_matrix():
    """
    according to
        https://stackoverflow.com/questions/57507832/unable-to-allocate-array-
with-shape-and-data-type

    run:
        echo 1 > /proc/sys/vm/overcommit_memory
    """
    vectorizer = CountVectorizer()
    transformer = TfidfTransformer()
    tf_idf = transformer.fit_transform(vectorizer.fit_transform(raw_corpus))

    weight = tf_idf.toarray()
    print(weight)
    return weight


def dimension_PCA(weight, dimension):
    pca = PCA(n_components=dimension)
    X = pca.fit_transform(weight)
    print(X)
    return X
```

```python
if __name__ == '__main__':
    # print(remove_stop_words(['首先', '你好', '难道说']))
    # preheat_cache("comments_zh")  # comments_zh
    weight = text_to_matrix()
    dimension_PCA(weight, dimension=800)
    # print(raw_corpus)
```

第一次运行结果

点击下载

第二次运行结果



```
C:\ProgramData\Anaconda3\envs\spider1\python.exe D:/diplomaProject/code/words_to_Matrix/toolkit.py
Building prefix dict from the default dictionary ...
Loading model from cache C:\Users\magic\AppData\Local\Temp\jieba.cache
Loading model cost 0.595 seconds.
Prefix dict has been built successfully.
add new words finished
cost 72.64095449447632
182583

飞黄腾达 10

欢天喜地 10

步步高升 10

Process finished with exit code 0
```

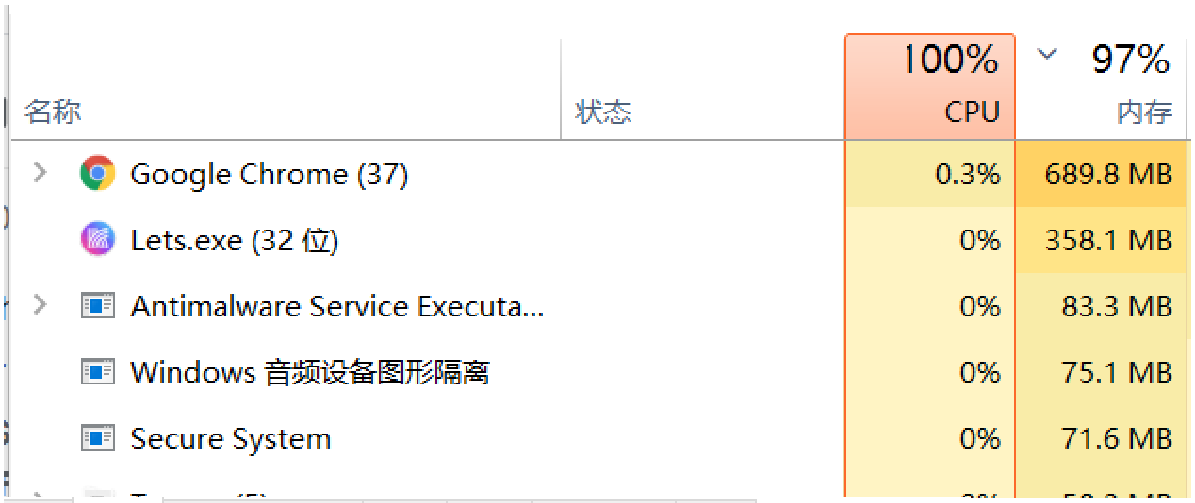
结果总结

语料库会随着爬取评论的增多而扩大，且收敛效果好。可以自适应不断扩大的评论数量。缺点是不能增量式的扩大，每次需要遍历整个语料库，存在重复计算，在服务器架构不优化的情况下，每次服务启动时初始化（预热阶段）的速度会不断变慢，虽然之后的响应速度几乎不变。

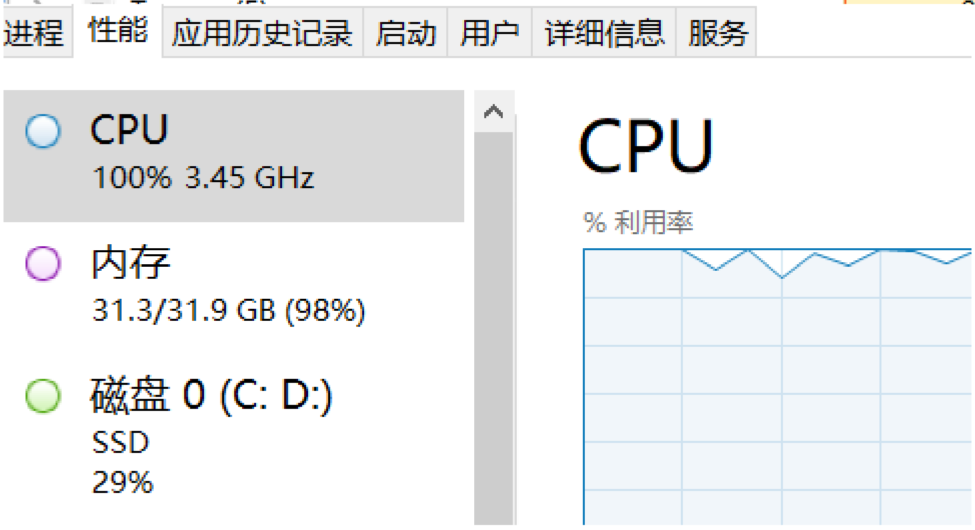
继续实现需求

```python
import redis
import json
import jieba
import jieba.analyse
import time
import torch
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.decomposition import PCA
from sklearn.cluster import Birch


corpus_file = "idf.txt.big"  # "idf.txt.big"
stop_words = open("stopwords.txt", encoding='utf-8').read()
corpus_fp = open(corpus_file, 'a+', encoding='utf-8')
raw_corpus = open(corpus_file, encoding='utf-8').readlines()
topK = 10


def remove_stop_words(data_list):
    word_list = []
    for data in data_list:
        if data not in stop_words:
            word_list.append(data)
    return word_list


def preheat_cache(db, sensitivity=4):
    """
    preheat 分词引擎
    add new words if discovered from comments,
    everytime service started, traversal the database, add new words to cache
    takes long time...
    wish astringency
    """
    pool = redis.ConnectionPool(host='localhost', port=6379,
decode_responses=True)
    r = redis.Redis(host='localhost', port=6379, decode_responses=True)
    jieba.analyse.set_idf_path(corpus_file)
    raw_comment = r.hgetall(db)
    raw_new_words_hash_map = {}
    start_time1 = time.time()
    for key, raw_value in raw_comment.items():
        value = json.loads(raw_value)
        comment_text = value['comment_text']
        comment_array = comment_text.split('|')
        comment = []
        for text in comment_array:
            # comment.append('/'.join(jieba.cut(text)))
            comment.append('/'.join(jieba.analyse.extract_tags(text,
topK=topK)))
        comment_row = '/'.join(comment)
        no_stop_words = remove_stop_words(comment_row.split('/'))
        for word in no_stop_words:
            if word in raw_new_words_hash_map:
                raw_new_words_hash_map[word] = raw_new_words_hash_map[word] + 1
            else:
                raw_new_words_hash_map[word] = 0
            jieba.add_word(word)
    print('add new words finished')
    new_words_hash_map = {}
    for key_word, times in raw_new_words_hash_map.items():
        if times >= sensitivity:
            new_words_hash_map[key_word] = times
    end_time1 = time.time()
    print("cost " + str(end_time1 - start_time1))
    append_corpus(new_words_hash_map)


def append_corpus(new_words):
    corpus = set()
    size = len(new_words)
    for item in raw_corpus:
        w = item.split(' ')[0]
        corpus.add(w)
    print(len(corpus))
    for key_word, times in new_words.items():
        if key_word not in corpus and len(key_word) > 2:
                new_item = '\n' + key_word + " " + str(topK)  # str(times/size)
                print(new_item)
                corpus_fp.write(new_item)


def text_to_matrix():
    vectorizer = CountVectorizer()
    transformer = TfidfTransformer()
    tf_idf = transformer.fit_transform(vectorizer.fit_transform(raw_corpus))

    weight = tf_idf.toarray()
    print(weight)
    return weight


def dimension_PCA(weight, dimension):
    pca = PCA(n_components=dimension)
    X = pca.fit_transform(weight)
    return X


def cluster_birch(X):
    brc = Birch(n_clusters=None)
    brc.fit(X)
    return brc.predict(X)


if __name__ == '__main__':
    # weight = text_to_matrix()
    # print(len(weight))
    # print(len(weight[0]))
    # fwt = torch.tensor(weight)
    fwt = torch.rand(1186667, 30)
    U, S, V = torch.pca_lowrank(fwt, q=30, center=True, niter=2)
    print(U)
    print(S)
    print(V)
    # with open("U.txt", 'w+') as f:
    #     f.write(str(U))
    # with open("S.txt", 'w+') as f:
    #     f.write(str(S))
    # with open("V.txt", 'w+') as f:
    #     f.write(str(V))
```

数据模型过大，ubuntu 虚拟机基本上一跑就卡死，把真机上的内存杀手 chrome 也挤得够呛。VScode 运行无响应，还是用命令行运行保险。



名称	状态	100% CPU	97% 内存
Google Chrome (37)		0.3%	689.8 MB
Lets.exe (32 位)		0%	358.1 MB
Antimalware Service Executa...		0%	83.3 MB
Windows 音频设备图形隔离		0%	75.1 MB
Secure System		0%	71.6 MB



进程 性能 应用历史记录 启动 用户 详细信息 服务

CPU
100% 3.45 GHz

内存
31.3/31.9 GB (98%)

磁盘 0 (C: D:)
SSD
29%

CPU
% 利用率

坑

> 我再也不用 Manjaro 了

Redis 迁移

1. `127.0.0.1>save` 后去对应目录下复制 `dump.rdb`
2. 若不知道要复制 到|从 哪个目录，可用 `127.0.0.1>config get dir` 获取
3. 覆盖|粘贴 `dump.rdb` 前一定要 `systemctl stop redis` 后 `ps -ef | grep redis` 看一下进程在不在，网上哪些用 `kill -9` 的杀死劲，不管用，杀了以后它换个 pid id 继续皮干。
4. `sudo cp` 一下，启动 redis，理应成功。

内存不够

unable to allocate gib for an array with shape

> 待了解原理？？？

> `numpy.core._exceptions.MemoryError: Unable to allocate array with shape` 这个报错windows下改了虚拟内存，pycharm下相应的设置也改了，都没用。所以只能去linux上重新开始项目了。`sudo -i` 切换 `root#` 执行 `echo 1 > /proc/sys/vm/overcommit_memory` 即可。

换成 linux 环境后执行上述命令

Process finished with exit code 137 (interrupted by signal 9: SIGKILL

用 pytorch 分批训练

用 chunk 切分数据仍然不行。。。

估摸着是读矩阵一次性读到内存中后继续在内存中操作会使内存占用变大然后超过门限，`torch.utils.data` 虽然可以分批加载数据，但是从硬盘中读。

存到硬盘中一端端读

将数据以不同的方式取部分，最后形成多个模型。可以以人工根据这些模型的好坏评个分，然后以分数为权重进行决策。或之间都赋一个分。

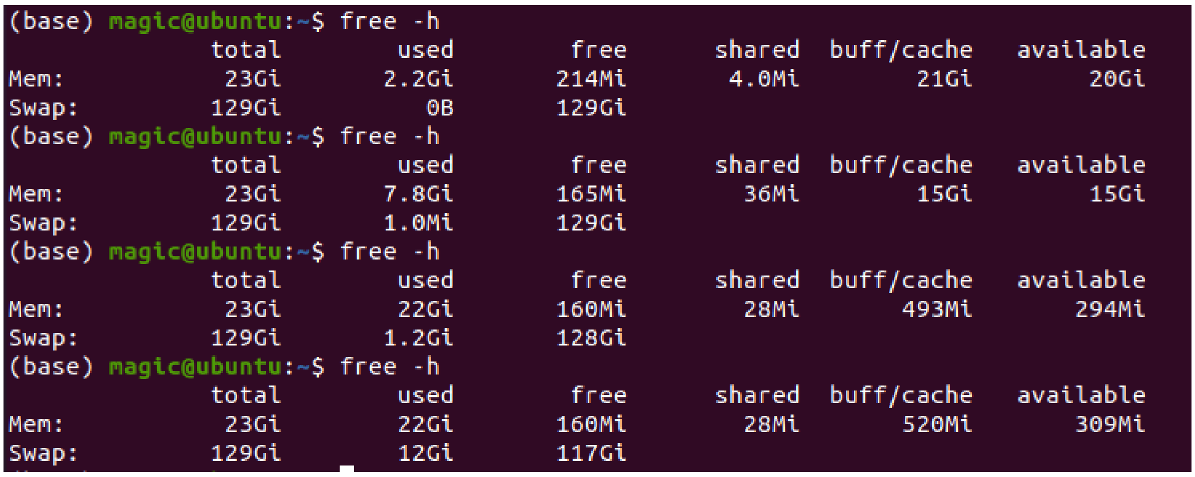
linux 扩容内存，虚存

> `free -m` 发现虚存不断扩大直到门限然后 killed 或 出现类似 `RuntimeError: [enforce fail at CPUAllocator.cpp:65] . DefaultCPUAllocator: can't allocate memory: you tried to allocate 127392453696 bytes. Error code 12 (Cannot allocate memory)` 的错误, 说明 内存虚存 分配的不够多

参考1，2, 在这里回答了

```
# disable the use of swap
sudo swapoff -a

# create the SWAP file. Make sure you have enough space on the hard disk.
sudo dd if=/dev/zero of=/swapfile bs=1024 count=136314880 status=progress
# 输出:
# 139458259968 bytes (139 GB, 130 GiB) copied, 472 s, 295 MB/s
# 136314880+0 records in
# 136314880+0 records out
# 139586437120 bytes (140 GB, 130 GiB) copied, 472.372 s, 296 MB/s

# Mark the file as SWAP space:
sudo mkswap /swapfile
# 输出:
# Setting up swapspace version 1, size = 130 GiB (139586433024 bytes)
# no label, UUID=25a565d9-d19c-4913-87a5-f02750ab625d

# enable the SWAP.
sudo swapon /swapfile

# check if SWAP is created
sudo swapon --show
# 输出:
# NAME      TYPE SIZE USED PRIO
# /swapfile file 130G   0B   -2

# Once everything is set, you must set the SWAP file as permanent, else you will lose the SWAP after reboot. Run this command:
echo '/swapfile none swap sw 0 0' | sudo tee -a /etc/fstab
```

之后看虚存有没有吃上,有没有用完



```
(base) magic@ubuntu:~$ free -h
              total        used        free      shared  buff/cache   available
Mem:           23Gi       2.2Gi       214Mi       4.0Mi        21Gi        20Gi
Swap:         129Gi          0B       129Gi
(base) magic@ubuntu:~$ free -h
              total        used        free      shared  buff/cache   available
Mem:           23Gi       7.8Gi       165Mi        36Mi        15Gi        15Gi
Swap:         129Gi       1.0Mi       129Gi
(base) magic@ubuntu:~$ free -h
              total        used        free      shared  buff/cache   available
Mem:           23Gi        22Gi       160Mi        28Mi       493Mi       294Mi
Swap:         129Gi       1.2Gi       128Gi
(base) magic@ubuntu:~$ free -h
              total        used        free      shared  buff/cache   available
Mem:           23Gi        22Gi       160Mi        28Mi       520Mi       309Mi
Swap:         129Gi        12Gi       117Gi
```

内存与CPU都成为速度瓶颈

前面的方法我都试过了，无奈数据集太大，就到了这里。

将原本几十万条数据取出1000条进行训练，还是到了这一步，也就是如果将数据分批计算的化到毕业我的毕设是做不完的。

如果租厉害的服务器又要花很多钱，又没得报销，坚决不干。

解决方案初步构想：

> 资源不够，算法来凑

那几十万维向量对应的词先自己分成`n`类，将维度降至`n`维，再对那`n`维进行PCA降维到`m`维(m <= n)，得到`几十万行(可不断增加，根据资源自适应) * m`的矩阵，再进行聚类。

参数说明
bs= sets the blocksize, for example bs=1M would be 1MiB blocksize.

count= copies only this number of blocks (the default is for dd to keep going forever or until the input runs out). Ideally blocks are of bs= size but there may be incomplete reads, so if you use count= in order to copy a specific amount of data (count*bs), you should also supply iflag=fullblock.

seek= seeks this number of blocks in the output, instead of writing to the very beginning of the output device.

最终解决方案

> 遇到的问题是：以具体的词语构成向量的维度形成的稀疏矩阵数据规模超过了我的硬件和时间承受能力。

使用 LTP4 进行对自然语言的词性划分，将原本几十万维的具体的词语构成向量转变成以词性构成的低维向量，虽然降低了训练的精度，但是硬件可以承受。

Linux 空间用完不要关机，保存之前写的代码，再继续操作

> 快照不频繁，一夜回到解放前

再次划分任务(正在进行的工作，之后要做的实现的需求)

- [ ] 使用 LTP 将 redis 中的评论逐条词性分词与词性标注，存于新建立的数据库。保证之后可以根据向量找到对应的评论
- [ ] 将新生成的向量拿 dbscan 聚类，若速度太慢则先用 PCA 降维，将输出的分类与对应的评论文本对应，用 真正的人工智能 判断分类的好坏。

- [ ] 调参，优化算法
- [ ] 将油猴端获取的评论用训练好的模型分类(将新增的评论加入聚类模型，后台定期优化模型)
- [ ] 将分类的结果传回油猴端对评论区进行删改

实现坑后需求

```
import numpy as np
from ltp import LTP
import redis
import json
from sklearn.cluster import Birch
from sklearn.cluster import DBSCAN

raw_comments_db = "tmp1"

pool = redis.ConnectionPool(host='localhost', port=6379, decode_responses=True)
r = redis.Redis(host='localhost', port=6379, decode_responses=True)


class IterateComments:
    def __init__(self, raw_comments):
        self.iter_raw = iter(raw_comments.items())

    def iter_loads_comments(self):
        try:
            value = json.loads(next(self.iter_raw)[1])
            return {  # 'comment_id': value['comment_id'], 'user_id':
value['user_id'], 'url_id': value['url_id'],
                'comment_text': value['comment_text'], 'comment_emoji':
value['comment_emoji'],
                'like_count': value['like_count'], 'reply_count':
value['reply_count'],
                'be_co_retweet': value['be_co_retweet'], 'be_co_comments':
value['be_co_comments'],
                'be_co_like': value['be_co_like'], 'be_contents':
value['be_contents'], 'be_emoji': value['be_emoji']}
        except StopIteration:
            return False
            # sys.exit()


class ProjectComments:
    def __init__(self):
        self.ltp = LTP()
        """
        这个词性标注官方文档不全的
https://ltp.readthedocs.io/zh_CN/latest/appendix.html
        那个代码中有但文档中没有的 'z' 不知道表示什么东东
        """
        self.switch = {'a': 0, 'b': 1, 'c': 2, 'd': 3, 'e': 4, 'g': 5, 'h': 6,
'i': 7, 'j': 8, 'k': 9, 'm': 10, 'n': 11,
                       'nd': 12, 'nh': 13, 'ni': 14, 'nl': 15, 'ns': 16, 'nt':
17, 'nz': 18, 'o': 19, 'p': 20, 'q': 21,
                       'r': 22, 'u': 23, 'v': 24, 'wp': 25, 'ws': 26, 'x': 27,
'z': 28}
```

```python
    def iter_project_quad(self, comment_text):
        splited = comment_text.split('|')
        return comment_text, self.part_of_speech(splited),
self.semantic_role(splited), \
                self.semantic_dependency_relations(splited),
self.semantic_relationship(splited)

    def part_of_speech(self, splited_comment_text):
        seg, hidden = self.ltp.seg(splited_comment_text)
        pos = self.ltp.pos(hidden)
        tag_list = [0] * 30
        for sub_vector in pos:
            for tag in sub_vector:
                tag_list[self.switch.get(tag, 29)] += 1
        return tag_list

    def semantic_role(self, comment_text):
        pass

    def semantic_dependency_relations(self, comment_text):
        pass

    def semantic_relationship(self, comment_text):
        pass


def cluster_birch(X, n_clusters=None):
    brc = Birch(n_clusters=n_clusters)
    brc.fit(X)
    return brc.predict(X)

def cluster_dbscan(X, n_cluster=3):
    dbs = DBSCAN()
    dbs.fit(X)
    return dbs.fit_predict(X)

if __name__ == '__main__':
    raw_comments = r.hgetall(raw_comments_db)
    pj_module = ProjectComments()
    iter_comment = IterateComments(raw_comments)
    ll = []
    while True:
        misc = iter_comment.iter_loads_comments()
        if not isinstance(misc, bool):
            ct = pj_module.iter_project_quad(misc['comment_text'])[1]
            ll.append(ct)
        else:
            break
            # print(type(misc['comment_text']))
        # if isinstance(misc, tuple):  # pj_module.iter_project_quad()
        #     ll.append(pj_module.iter_project_quad(misc)[1])
        # else:
        #     break
    # print(np.array(ll))
    print(cluster_dbscan(np.array(ll)))
    print(cluster_birch(np.array(ll)))
```

将网页评论传到后端

划分任务

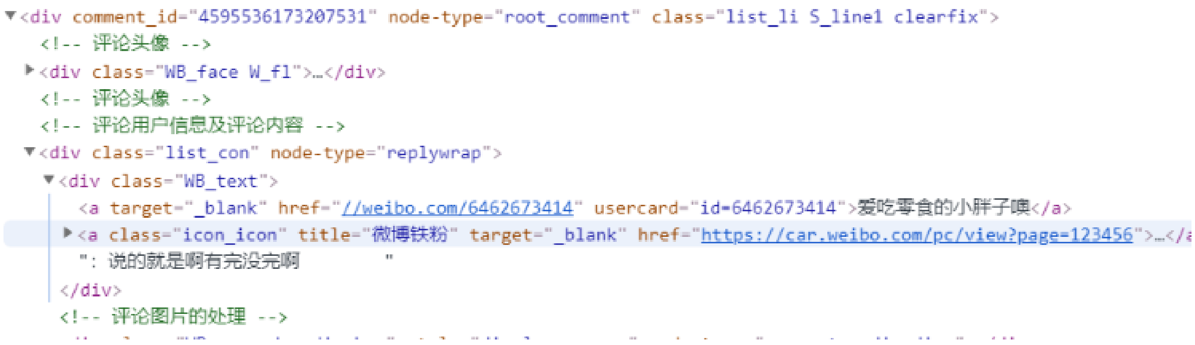
- [x] js脚本抓取评论
- [ ] node.js 与 python 交互
- [ ] 将评论向量化，分类
- [ ] 根据评论效果调整
- [ ] 前后端交互，将对应无聊的评论删除

js脚本抓取评论

选一个业务场景

选一个含微博评论的页面 https://weibo.com/5639089198/JE5k6rgXV?type=comment

分析源码



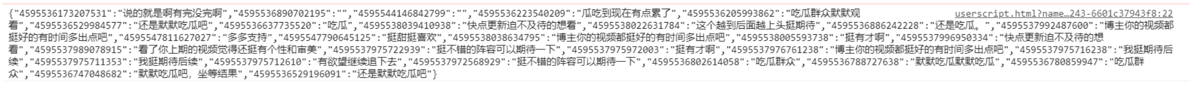
```
▼<div comment_id="4595536173207531" node-type="root_comment" class="list_li S_line1 clearfix">
   <!-- 评论头像 -->
  ▶<div class="WB_face W_fl">…</div>
   <!-- 评论头像 -->
   <!-- 评论用户信息及评论内容 -->
  ▼<div class="list_con" node-type="replywrap">
    ▼<div class="WB_text">
       <a target="_blank" href="//weibo.com/6462673414" usercard="id=6462673414">爱吃零食的小胖子噢</a>
      ▶<a class="icon_icon" title="微博铁粉" target="_blank" href="https://car.weibo.com/pc/view?page=123456">…</a
       "：说的就是啊有完没完啊          "
     </div>
     <!-- 评论图片的处理 -->
```

编写脚本

> 获取评论，并转为便于网络传输的 json 格式，打印，检验。

```
function sleep(ms) {
    return new Promise(resolve => setTimeout(resolve, ms));
}
(async() => {
    'use strict';
    await sleep(3000); // 之后迭代忙等监控网页是否加载出来

    // 爬一级评论
    var data = [].slice.call(document.querySelectorAll("[node-
type='root_comment']")).map(function(ele) {
        let commentId = ele.getAttribute("comment_id"),
            text = ele.getElementsByClassName("WB_text")[0].lastChild;
        return [commentId, text.wholeText.substring(1).trim()];
    }); //部分长的评论没有爬下来

    // convert array to Json
    var obj = {};
    data.forEach(ele => obj[ele[0].toString()] = ele[1]);
    console.log(JSON.stringify(obj));
})()
```

效果



```
{"4595536173207531":"说的就是啊有完没完啊","4595536890702195":"","4595544146842799":"","4595536223540209":"瓜吃到现在有点累了","4595536205993862":"吃瓜群众默默观          userscript.html?name…243-6601c37943f8:22
看","4595536529984577":"还是默默吃瓜吧","4595536637735520":"吃瓜","4595538039410938":"快点更新迫不及待的想看","4595538022631784":"这个越到后面越上头挺期待","4595536886242228":"还是吃瓜。","4595537992487600":"博主你的视频都
挺好的有时间多出点吧","4595547811627027":"多多支持","4595547790645125":"挺甜挺喜欢","4595538038634795":"博主你的视频都挺好的有时间多出点吧","4595538005593738":"挺有才啊","4595537996950334":"快点更新迫不及待的想
看","4595537989078915":"看了你上期的视频觉得还挺有个性和审美","4595537975722939":"挺不错的阵容可以期待一下","4595537975972003":"挺有才啊","4595537976761238":"博主你的视频都挺好的有时间多出点吧","4595537975716238":"我挺期待后
续","4595537975711353":"我挺期待后续","4595537975712610":"有欲望继续追下去","4595537972568929":"挺不错的阵容可以期待一下","4595536802614058":"吃瓜群众","4595536788727638":"默默吃瓜默默吃瓜","4595536780859947":"吃瓜群
众","4595536747048682":"默默吃瓜吧，坐等结果","4595536529196091":"还是默默吃瓜吧"}
```

前后端通信

前端送数据到后端

服务端

```
const express = require('express')
const bodyParser = require('body-parser')
app = express()

port = 3000

const cors = require('cors')
var jsonParser = bodyParser.json()
app.use(bodyParser.urlencoded({ extended: false }))
var corsOptions = {
    origin: 'https://weibo.com',
    optionsSuccessStatus: 200 // some legacy browsers (IE11, various SmartTVs)
choke on 204
}
app.listen(port, () => {
    console.log('port is listening')
})
app.options('*', cors())
app.post('/simple-cors', jsonParser, cors(), (req, res) => {
    res.header('Access-Control-Allow-Origin', req.headers.origin || "*");
    res.header('Access-Control-Allow-Methods',
'GET,POST,PUT,HEAD,DELETE,OPTIONS');
    res.header('Access-Control-Allow-Headers', 'content-Type,x-requested-with');
    console.info("POST /simple-cors");
    console.log(req.body);
    res.json({
        text: "Simple CORS requests are working. [POST]"
    })
})
```

油猴端

```
const xhr = new XMLHttpRequest()
xhr.onreadystatechange = () => {
    if(xhr.readyState === 4 && xhr.status === 200){
        console.log(xhr.responseText)
    }
}
xhr.open('POST', 'http://127.0.0.1:3000/simple-cors')
xhr.setRequestHeader("Content-type", "application/json")
xhr.send(data)
```

经验

- 将数据从网页上爬下来不要过分清洗，从而破坏原来的结构，要记录爬取每条数据时的时间。要在尽可能小的占用存储空间的基础上尽可能多的保存数据之间的关系。
- 能利用的资源（账号，IP，服务器~~，其实就是钱~~）越多，个人的潜力与才能才会得到更大发展。真的是马太效应啊！
- 框架或现有的设计模式 既是模板，也是束缚，束缚对应已经限定死的场景，稍微活一点的场景就要用本真屎山了。
- 本人日相属马，相五行属火，适合互联网行业，但切忌完美主义。
- 做项目运用的新知识跟的后继新知识结点不能太远，比如刚学了Docker还没用熟到场景适配（不是为了用而用，是为了更方便解决场景的需求而用。虽然可能有更好的方法，但这个方法的技能里我目前的技能树还比较远，那么宁可用离它比较近的还没熟练的方法（走出舒适区，远离危险区））就不要继续学ansible，最后学了一大堆却一事无成，但自以为很NB，以QQ群里那些走火入魔到不屑于去打工的人为戒。
- Windows上写东西遇到问题提前报错，Linux上遇到问题可以修改系统，然后遇到的问题就更多了。而且，就我目前的技术栈来说，就开发而言，Windows可以满足需求。